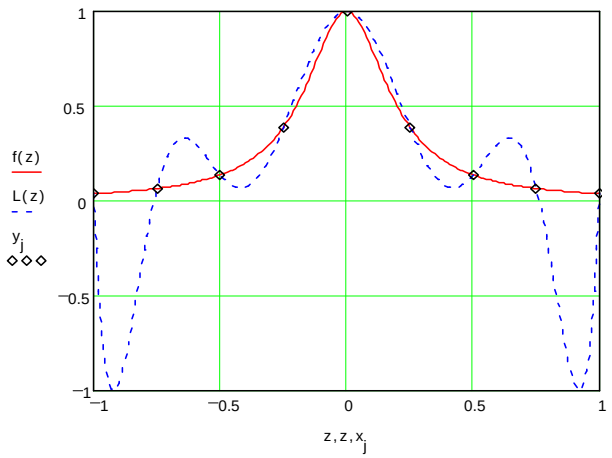
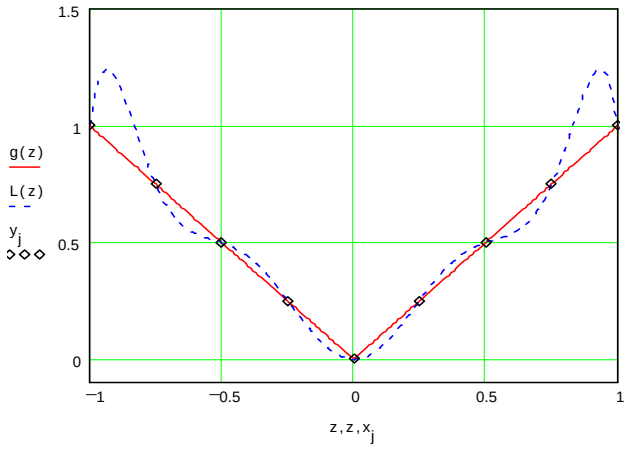
Material didactic pentru orele de laborator

Maria CAPCELEA, Titu CAPCELEA, Ion SECRIERU

METODE ȘI ALGORITMI LA ANALIZA NUMERICĂ





UNIVERSITATEA DE STAT DIN MOLDOVA

FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ

CATEDRA MATEMATICA APLICATĂ

MARIA CAPCELEA, TITU CAPCELEA, ION SECRIERU

METODE ŞI ALGORITMI LA ANALIZA NUMERICĂ

Aprobat şi recomandat de
Consiliul Facultății de
Matematică şi Informatică

Chişinău, 2009
CEP USM

CZU 519.6 (075.8)
S

Recomandat de Catedra Matematica Aplicată

Recenzent **Boris HÂNCU**, dr., conf. univ.

Descrierea CIP a Camerei Naționale a Cărții
Capcelea, Maria
Metode şi algoritmi la Analiza Numerică / Maria Capcelea, Titu Capcelea, Ion Secrieru; Universitatea de Stat din Moldova. Facultatea de Matematică şi Informatică. Catedra Matematica Aplicată. – Chişinău: CEP USM, 2009. – 73p.
Bibliogr. p. 72 (28 titluri)
ISBN 9975-70-900-2
50 ex.

519.6 (075.8)

© Maria Capcelea, Titu Capcelea, Ion Secrieru, 2009
© USM, 2009

ISBN 9975-70-900-2

CUPRINS

INTRODUCERE……..... 5
LUCRAREA DE LABORATOR №1
Aproximarea valorilor funcțiilor cu ajutorul dezvoltării în serii de puteri.....….....7
1.I. Scopul lucrării.....……..……...7
1.II. Prezentarea teoretică a lucrării.....…….....…... 7
♦ Formularea problemei.....……...…… 7
♦ Descrierea algoritmului..... 7
1.III. Modul de desfăşurare a lucrării.....11
♦ Cerințe față de îndeplinirea lucrării.....……...……11
♦ Chestiuni de studiat.....11
♦ Probleme propuse spre rezolvare.....11
LUCRAREA DE LABORATOR №2
Aproximarea funcțiilor cu polinoame de interpolare.....….....13
2.I. Scopul lucrării.....……..……...13
2.II. Prezentarea teoretică a lucrării.....…….....…... 13
♦ Formularea problemei.....……...…… 13
♦ Interpolare polinomială pe rețea neuniformă.....13
Polinomul de interpolare sub forma lui Lagrange..... 13
Algoritmul lui Neville-Aitken.....15
Polinomul de interpolare sub forma lui Newton cu diferențe divizate..... 16
♦ Interpolare polinomială pe rețea uniformă..... 18
Polinomul de interpolare sub forma lui Lagrange cu noduri echidistante..... 18
Polinomul de interpolare sub forma lui Newton cu noduri echidistante.....19
A) Formula lui Newton de interpolare ascendentă.....19
B) Formula lui Newton de interpolare descendentă.....21
2.III. Modul de desfăşurare a lucrării.....22
♦ Cerințe față de îndeplinirea lucrării.....……...……22
♦ Chestiuni de studiat.....23
♦ Probleme propuse spre rezolvare.....23
LUCRAREA DE LABORATOR №3
Aplicarea formulelor de cuadratură la calculul aproximativ al integralei definite......….....24
3.I. Scopul lucrării.....……..……...24
3.II. Prezentarea teoretică a lucrării.....…….....…... 24
♦ Formularea problemei.....……...…… 24
♦ Formula dreptunghiurilor…….....25
♦ Formula trapezelor……..…….....25
♦ Formula lui Simpson (parabolelor).…….....25
♦ Algoritmi cu control automat al pasului de integrare..... 26
♦ Formule de cuadratură de tip Gauss...…….....29
Formula de cuadratură a lui Gauss-Legendre..... 30
Formula de cuadratură a lui Gauss-Chebyshev.....31
3.III. Modul de desfăşurare a lucrării.....31
♦ Cerințe față de îndeplinirea lucrării.....……...……31
♦ Chestiuni de studiat.....32
♦ Probleme propuse spre rezolvare.....32
LUCRAREA DE LABORATOR №4
Rezolvarea numerică a ecuațiilor algebrice şi transcendente.....….....33
4.I. Scopul lucrării.....……..……...33
4.II. Prezentarea teoretică a lucrării.....…….....…... 33
♦ Formularea problemei.....……...…… 33
♦ Separarea rădăcinilor prin metoda tabelării funcției..….....33

♦ Separarea grafică a rădăcinilor ……..…..... 34
♦ Metoda dihotomiei……..…….....34
♦ Metoda lui Newton…….....35
♦ Metoda lui Newton modificată……..…….....37
♦ Metoda secantelor (Newton discretă)……..…...38
♦ Metoda coardelor (interpolării liniare, falsei poziții, secantelor cu un capăt fixat)……39
♦ Metoda iterației simple……... 41
♦ Calculul rădăcinilor ecuațiilor polinomiale…… 42
♦ Calculul rădăcinilor multiple…….. 43
4.III. Modul de desfăşurare a lucrării.....43
♦ Cerințe față de îndeplinirea lucrării.....……...……43
♦ Chestiuni de studiat.....44
♦ Probleme propuse spre rezolvare.....44
LUCRAREA DE LABORATOR №5
Rezolvarea numerică a sistemelor de ecuații algebrice liniare.....…..... 46
5.I. Scopul lucrării.....……...……...46
5.II. Prezentarea teoretică a lucrării.....……...….. 46
♦ Formularea problemei.....……...…… 46
♦ Metode directe……...…..... 46
Metoda eliminării a lui Gauss…… 46
Strategii de pivotare în rezolvarea SEAL…….. 48
Aplicații ale algoritmilor de eliminare……...49
A) Rezolvarea SEAL cu matrice rare……49
B) Calculul determinantului şi a inversei matricei A……50
Metoda factorizării LU……...51
Metoda factorizării Doolittle……51
Metoda factorizării Crout……..51
Metoda factorizării Cholesky……52
♦ Metode iterative……..…..... 55
Metode staționare……...55
Metoda lui Jacobi…..…….....…… 55
Metoda lui Gauss-Seidel……..…… 56
O metodă de stocare compactă a matricei simetrice…….…… 58
Metode nestaționare……...59
Metoda pasului descendent (coborârii rapide)...……...…… 59
Metoda gradienților conjugați..…… 61
5.III. Modul de desfăşurare a lucrării.....63
♦ Cerințe față de îndeplinirea lucrării.....……...……63
♦ Chestiuni de studiat.....63
LUCRAREA DE LABORATOR №6
Metoda lui Runge-Kutta de rezolvare a problemei lui Cauchy.....…..... 64
6.I. Scopul lucrării.....……...……...64
6.II. Prezentarea teoretică a lucrării.....……...….. 64
♦ Formularea problemei.....……...…… 64
♦ Descrierea algoritmilor...……...…..... 64
♦ Monitorizarea erorilor locale de trunchiere pe parcursul procesului de integrare.....66
♦ Algoritmi cu controlul automat al pasului…….....…... 67
6.III. Modul de desfăşurare a lucrării.....70
♦ Cerințe față de îndeplinirea lucrării.....……...……70
♦ Chestiuni de studiat.....71
♦ Probleme propuse spre rezolvare.....71
REFERINȚE.....72

INTRODUCERE

Aplicarea tehnicii computaționale în cele mai diverse domenii ale activității umane astăzi a devenit un lucru obişnuit. Însă utilizarea eficientă a calculatorului la rezolvarea problemelor aplicative presupune identificarea unor metode de rezolvare a acestora, descrierea lor într-un limbaj algoritmic, apoi implementarea şi testarea pe calculator. Metodele matematice ce stau la baza algoritmilor respectivi sunt studiate în cadrul cursului *Analiza numerică*, disciplină obligatorie în planurile de învățământ la toate specialitățile ce țin de *Matematica aplicată* şi *Informatică*.

Lucrarea de față se adresează, în primul rând, studenților din învățământul universitar pentru a servi ca material didactic la îndeplinirea lucrărilor de laborator prevăzute în procesul de studiu al cursului *Analiza numerică*. Menirea lor este de a contribui la însuşirea mai profundă a metodelor de rezolvare aproximativă a problemelor matematice, cât şi a formării gândirii algoritmice, deprinderilor de programare şi analiză a rezultatelor obținute.

În lucrare sunt incluse lucrări de laborator ce se referă la următoarele compartimente: metode numerice de aproximare a funcțiilor, construirea şi aplicarea formulelor de cuadratură, metode aproximative de rezolvare a ecuațiilor neliniare, metode exacte şi aproximative de rezolvare a sistemelor de ecuații liniare algebrice, metode de rezolvare aproximativă numerică a problemei lui Cauchy. Compartimentele menționate corespund Curriculumului disciplinei *Analiza numerică* pentru specialitățile matematice, cât şi cursului *Calcul numeric* pentru specialitățile informatice.

Materialul este structurat astfel, încât fiecare lucrare conține trei compartimente. Anume, în partea I sunt formulate scopul şi obiectivele lucrării. Partea a II-a conține formularea strictă a problemei şi metodele principale de rezolvare a acesteia cu formulele respective de calcul şi de estimare a erorii. Pentru majoritatea metodelor, într-o secțiune specială, se face o analiză a proprietăților numerice pe care le posedă. Ultimul compartiment include cerințele principale de îndeplinire a lucrărilor de laborator şi un set de variante de probleme concrete matematice pentru fiecare lucrare.

O atenție deosebită se acordă aspectelor algoritmice şi organizării procesului de calcul. Astfel, metodele numerice, descrise în lucrare, sunt concretizate sub formă de algoritmi, prezentați într-un meta-limbaj extrem de apropiat de implementarea directă într-un limbaj de programare de nivel înalt.

Menționăm faptul că în prezent există sisteme mai performante de utilizare a metodelor aproximative de rezolvare a problemelor matematice studiate în această lucrare, cât şi ale altor metode din *Analiza numerică*. Însă aspectul programării nemijlocite a metodelor pe probleme

concrete considerăm că este o etapă absolut necesară la pregătirea specialiştilor de o calificare înaltă în domeniul matematicii aplicate.

În lucrare prezentăm şi câteva informaţii, utile pentru studenţi, referitoare la modul de desfăşurare a laboratorului la *Analiza numerică*. Fiecare lecţie de laborator are o anumită tematică. Activitatea pe care trebuie să o desfăşoare fiecare student în cadrul lecţiei presupune două părţi importante. Prima parte constă în exploatarea unor programe demonstrative ce ilustrează tematica lucrării. A doua parte a lecţiei constă în implementarea unora dintre algoritmii studiaţi în cadrul temei respective. Implementarea se va face într-un limbaj de programare de nivel înalt, în concordanţă cu pseudocodul prezentat în lucrare. În urma analizei comportamentului numeric al algoritmilor implementaţi, studentul trebuie să redacteze un referat, conform cerinţelor fiecărei lucrări, pe care îl va prezenta profesorului în momentul susţinerii lucrării de laborator.

Fiecare temă de laborator este apreciată cu două note: o notă pentru referat şi o notă pentru implementarea algoritmilor studiaţi în lucrarea respectivă într-un limbaj de programare de nivel înalt. Referatul va conţine următoarele compartimente: numele studentului şi grupa din care face parte; numele cadrului didactic îndrumător; titlul lucrării, scopul lucrării, principiul algoritmilor, pseudocodul algoritmilor, rezultate experimentale, grafice etc. (conform cerinţelor din îndrumar), analiza rezultatelor şi concluzii. În principiu, cerinţele pot fi modificate de către profesor.

În încheiere, autorii mulţumesc colegilor lor - dnei Galina Râbacova, dr., conf. univ., şi dlui Vladimir Paţiuc, dr., conf. univ., pentru interesul acordat şi în special pentru comentariile şi observaţiile constructive făcute pe marginea lucrării.

Autorii

LUCRAREA DE LABORATOR №1
APROXIMAREA VALORILOR FUNCȚIILOR CU AJUTORUL DEZVOLTĂRII ÎN SERII DE PUTERI

1.I. Scopul lucrării

Studiul unui algoritm pentru sumarea numerică a seriilor de puteri cu controlul erorii de trunchiere şi implementarea acestuia într-un limbaj de programare de nivel înalt.

1.II. Prezentarea teoretică a lucrării

♦ Formularea problemei

Funcția $y = f(x)$ se exprimă prin funcțiile elementare:

$$\cos x,\ \sin x,\ e^x \quad (x \in \mathbb{R}). \tag{1.1}$$

Pentru o valoare x_0 a argumentului x să se aproximeze cu exactitatea $\varepsilon > 0$ valorile funcțiilor indicate în relația (1.1), utilizând în acest scop formulele pentru dezvoltarea lor în serii de puteri, apoi, în baza valorilor obținute, să se calculeze valoarea aproximativă $y_0 = f(x_0)$ a funcției inițiale.

♦ Descrierea algoritmului

Dezvoltările în serii de puteri ale funcțiilor indicate în relația (1.1) sunt:

$$\cos x = \sum_{j=0}^{\infty} (-1)^j x^{2j} / (2j)!, \quad \sin x = \sum_{j=0}^{\infty} (-1)^j x^{2j+1} / (2j+1)!, \quad e^x = \sum_{j=0}^{\infty} x^j / j!, \quad x \in \mathbb{R}. \tag{1.2}$$

Reprezentările din relația (1.2) permit evaluarea valorilor funcțiilor menționate prin reducerea la operații aritmetice elementare (aditive şi multiplicative).

Calculul aproximativ al sumei unei serii presupune trunchierea unui proces de sumare infinit. Dacă şirul numeric $\{x_k\}_{k \geq 0}$ converge către limita x, atunci pentru orice $\varepsilon > 0$ există un număr natural n, astfel încât $|x - x_k| \leq \varepsilon$, pentru orice $k > n$. În consecință, pentru o exactitate impusă ε, există un rang finit n, astfel încât x_n reprezintă o aproximație satisfăcătoare pentru limita x. Rezultă că, după evaluarea unui număr n suficient de mare de termeni, ultimul poate fi adoptat ca soluție numerică cu o eroare de trunchiere dependentă de acest n.

Deoarece seriile de puteri corespunzătoare funcțiilor cosinus şi sinus sunt alternante, iar şirurile valorilor absolute ale termenilor sunt monoton descrescătoare, conform criteriului lui Leibniz, eroarea de trunchiere este majorată de modulul primului termen neglijat. În realitate, însă, erorile de rotunjire şi modul de sumare a seriei (de la stânga spre dreapta, fără a ține cont de mărimea fiecărui termen) pot genera o eroare de calcul mult mai mare decât cea indicată de criteriul lui Leibniz (a se vedea, de exemplu, [28, p.22]). De aceea, în procesele de calcul ale valorilor funcțiilor cu ajutorul dezvoltării în serii de puteri, este necesară întreprinderea unor măsuri speciale pentru a evita pierderi semnificative în exactitate.

Întrucât pentru eroarea absolută a mărimii x^j are loc relația $\Delta(x^j) = x^j \delta(x^j) = j\, x^j \delta(x)$ (la ridicarea la putere a unui număr aproximativ eroarea lui relativă se înmulțeşte la exponenta puterii), în cazul în care nu se îndeplineşte inegalitatea $|x| < 1$, eroarea absolută poate deveni inacceptabil de mare. Prin urmare,

influența erorilor de rotunjire se va diminua semnificativ dacă $|x|<1$.

Utilizând paritatea (imparitatea) funcțiilor cosinus şi sinus $\cos x=\cos|x|$, $\sin x=m\sin|x|$, $m=sign\,x$, dar şi egalitatea $e^x\cdot e^{-x}=1$, se reduce calculul valorilor funcțiilor din relația (1.1) pentru un argument arbitrar la calculul valorilor acestora pentru un argument nenegativ. Deoarece la sumarea seriilor menționate eroarea de calcul va fi esențial mai mică dacă $|x|<1$, e important a reduce calculul valorilor funcțiilor $\cos x$ şi $\sin x$ de argument nenegativ la argument $t\in[0;\pi/4]$, utilizând în acest scop formule de reducere, iar calculul valorilor funcției e^x - la argument $t\in[0;1)$. După această reducere sunt utilizate egalitățile definite de relația (1.2). În cele ce urmează, vom explica în detalii procesul de calcul al valorilor funcțiilor indicate în relația (1.1).

Pentru funcțiile $\cos x$ şi $\sin x$ reducerea menționată se efectuează consecutiv prin diminuarea argumentului până la valorile $x<2\pi$, $x<\pi$, $x<\pi/2$, $x<\pi/4$. În acest scop, este utilizată funcția $[x]$ ce calculează partea întreagă a argumentului, precum şi următoarele formule de reducere:

$$\cos(2\pi n+x)=\cos x;\ \cos(\pi\pm x)=-\cos x;\ \sin(2\pi n+x)=\sin x;\ \sin(\pi\pm x)=\mp\sin x\quad (n\in\mathbb{N}).$$

Dacă $x\in(\pi/4;\pi/2)$, atunci se efectuează reducerea argumentului până la mărimea $\pi/2-x$ în timp ce calculul cosinusului (sinusului) se reduce la calculul sinusului (cosinusului):

$$\cos(\pi/2-x)=\sin x;\ \sin(\pi/2-x)=\cos x.$$

De exemplu, pentru funcția $\cos x$ şi $x=7.6\pi$ $(m=1)$ efectuăm următoarele calcule:

$$n=[7.6\pi/2\pi]=[3.8]=3;\ x-2\pi n=1.6\pi>\pi;\ m=-1;\ x-\pi=0.6\pi>\pi/2;$$
$$\pi-x=0.4\pi>\pi/4;\ m=1;\ t=\pi/2-x=0.1\pi.$$

Prin urmare, $\cos 7.6\pi=\sin 0.1\pi$ şi în continuare se calculează sinusul conform formulei pentru dezvoltarea în serie de puteri.

Se presupune că argumentul este definit în radiani. Dacă acesta este definit în grade, atunci el se transformă în radiani prin înmulțirea cu mărimea $\pi/180$.

În calculul valorilor funcțiilor $\cos t$ şi $\sin t$ ($t\in(0;\pi/4]$; cazul $t=0$ este trivial şi în algoritm se descrie separat) cu exactitatea $\varepsilon>0$ se utilizează egalitățile corespunzătoare din relația (1.2), iar procesul de sumare se încheie când pentru un $j_*\in\mathbb{N}$ se îndeplineşte inegalitatea [24, p.99; 25] $|a_{j_*}|<\varepsilon$, unde $a_{j_*}=(-1)^{j_*}t^{2j_*}/(2j_*)!$ (pentru funcția cosinus) sau $a_{j_*}=(-1)^{j_*}t^{2j_*+1}/(2j_*+1)!$ (pentru funcția sinus). Sumarea seriei pentru funcția cosinus se face prin formulele recurente [25, p.29]: $S_0=1, a_0=1, a_j=-a_{j-1}t^2/((2j-1)2j)$, $S_j=S_{j-1}+a_j$, $j=1,2,\ldots$, iar a seriei pentru funcția sinus prin formulele: $S_0=t, a_0=t, a_j=-a_{j-1}t^2/(2j(2j+1))$, $S_j=S_{j-1}+a_j$, $j=1,2,\ldots$.

Pseudocodul algoritmului

Prezentăm algoritmii de calcul ai valorilor funcțiilor $\cos x$ şi $\sin x$ cu o eroare de trunchiere mai mică decât ε.

Date de intrare:
x_0 (de tip real) – punctul în care se calculează valoarea funcției $\cos x$ sau $\sin x$;

ε (de tip real) – exactitatea de calcul.
Date de ieşire:
x_0 (de tip real) – punctul în care se calculează valoarea funcției $\cos x$ sau $\sin x$;
s (de tip real) – valoarea funcției $\cos x$ sau $\sin x$ în punctul x_0;
k (de tip întreg) – numărul de termeni sumați;
real x, s
întreg k, m, n, p

(calculul valorii $\cos x_0$)	(calculul valorii $\sin x_0$)
dacă $\lvert x_0\rvert > \varepsilon$ atunci	dacă $\lvert x_0\rvert > \varepsilon$ atunci
{	{
$m := sign\, x_0;\ x := \lvert x_0\rvert$	$m := sign\, x_0;\ x := \lvert x_0\rvert$
dacă $x \ge 2\pi$ atunci $\{n := [x/2\pi];\ x := x - 2\pi n\}$	dacă $x \ge 2\pi$ atunci $\{n := [x/2\pi];\ x := x - 2\pi n\}$
dacă $x \ge \pi$ atunci $\{x := x - \pi;\ m := -m\}$	dacă $x \ge \pi$ atunci $\{x := x - \pi;\ m := -m\}$
dacă $x \ge \pi/2$ atunci $\{x := \pi - x;\ m := -m\}$	dacă $x \ge \pi/2$ atunci $x := \pi - x$
dacă $x \ge \pi/4$ atunci	dacă $x \ge \pi/4$ atunci
$\{x := \pi/2 - x;\ s := x;\ a := -x^3/6;\ p := 2\}$	$\{x := \pi/2 - x;\ s := 1;\ a := -x^2/2;\ p := 1\}$
altfel $\{s := 1;\ a := -x^2/2;\ p := 1\}$	altfel $\{s := x;\ a := -x^3/6;\ p := 2\}$
$k := 1$	$k := 1$
execută	execută
{	{
$s := s + a;\ k := k + 1$	$s := s + a;\ k := k + 1$
$p := p + 2;\ a := -ax^2/(p(p+1))$	$p := p + 2;\ a := -ax^2/(p(p+1))$
}	}
cât timp $\lvert a\rvert \ge \varepsilon$	cât timp $\lvert a\rvert \ge \varepsilon$
$s := ms$	$s := ms$
}	}
altfel $s := 1;\ k := 0$	altfel $s := x_0;\ k := 0$
scrie x_0, s, k	scrie x_0, s, k

Calculul valorii funcției e^x pentru $x \ge 0$ se efectuează în modul următor:

Fie n - partea întreagă a lui x şi $t = x - n$. Evident, $t \in [0;1)$. Dacă $t = 0$ şi $n = 0$, atunci $e^x = 1$, iar dacă $t = 0$ şi $n \ge 1$, atunci $e^x = e^n = \underbrace{e \cdot \ldots \cdot e}_{de\ n\ ori}$. Pentru $t \in (0;1), n \ge 0$ avem $e^x = e^n e^t$ şi e^t se calculează cu exactitatea $\varepsilon > 0$, conform formulei corespunzătoare din relația (1.2), utilizând suplimentar următoarea relație pentru restul seriei (a se vedea [24, p.89]) $e^t - S_p < a_p t/p$, $S_p = \sum_{j=0}^{p} a_j$, $a_j = t^j/j!$, $p = 1, 2, \ldots$. Astfel, sumarea se va efectua până la acea valoare p_* a lui p, pentru care se va satisface relația $a_{p_*} t/p_* < \varepsilon$. Procesul sumării este comod de organizat în modul următor [25, p.27]: $S_0 = 1, a_0 = 1, a_j = a_{j-1} t/j$, $S_j = S_{j-1} + a_j$, $j = 1, 2, \ldots$.

Pseudocodul algoritmului

Date de intrare:
x_0 (de tip real) – punctul în care se calculează valoarea funcției e^x;
ε (de tip real) – exactitatea de calcul.
Date de ieşire:
x_0 (de tip real) – punctul în care se calculează valoarea funcției e^x;
exp (de tip real) – valoarea funcției e^x în punctul x_0;
$k+1$ (de tip întreg) – numărul de termeni sumați;

```
întreg k, m, n
real x, t, exp, s, a
    m := sign x0; x := |x0|; n := [x]; t := x − n; exp := 1; s := 1
    pentru i := 1,n executată s := s · e
    dacă t ≠ 0 atunci
      {
        a := 1; k := 0
        executată
          {
            k := k + 1; a := a · t/k; exp := exp + a
          }
        cât timp at/k ≥ ε
      }
    exp := exp · s
    dacă m = −1 atunci exp := 1/exp
scrie x0, exp, k + 1
```

Întrucât funcțiile $tgx = \sin x/\cos x$, $ctgx = \cos x/\sin x$, $shx = (e^x - e^{-x})/2$, $chx = (e^x + e^{-x})/2$, $thx = shx/chx$ sunt dependente de funcțiile indicate în relația (1.1), se admite ca expresia analitică a lui $f(x)$ să conțină unele dintre acestea. Pentru a evita pericolul pierderilor în exactitate din cauza efectuării diferenței a două mărimi apropiate ca valoare, calculul valorilor funcțiilor $sh\,x$ şi $th\,x$ pentru valori $x \approx 0$ se va face în baza formulei $shx = \sum_{j=0}^{\infty} x^{2j+1}/(2j+1)!$ (în locul celei corespunzătoare funcției e^x). Totodată, pentru valorile $x < 0$ (dar apropiate de zero) se va ține cont şi de egalitatea $sh(-x) = -shx$.

Analiza algoritmului

Eroarea numerică comisă în calculul aproximativ al valorilor funcțiilor elementare din relația (1.1), utilizând dezvoltarea în serie de puteri, constă din două componente: eroarea de rotunjire (datorată reprezentării finite, aproximative a numerelor reale în sistemele de calcul) şi eroarea de trunchiere (datorată aproximării finite a unor procese infinite). În cazul când numărul de termeni ai seriei este fixat, eroarea de trunchiere va depinde direct proporțional de valoarea argumentului t.

Au loc următoarele estimații pentru eroarea de trunchiere a seriilor utilizate în lucrare [24, p.89; p.95]:

$$\left|\cos t - S_n^{\cos}\right| \le t^{2(n+1)}/(2(n+1))!,\ \left|\sin t - S_n^{\sin}\right| \le t^{2n+3}/(2n+3)!,\ t \in [0;\pi/4];\ \left|e^t - S_n^{\exp}\right| < t^{n+1}/(n!n),\ t \in [0;1],$$

unde $S_n^{\cos} = \sum_{j=0}^{n} (-1)^j t^{2j}/(2j)!$, $S_n^{\sin} = \sum_{j=0}^{n} (-1)^j t^{2j+1}/(2j+1)!$, $S_n^{\exp} = \sum_{j=0}^{n} t^j/j!$. Modul în care eroarea de trunchiere depinde de n caracterizează viteza de convergență a algoritmilor descrişi.

În procesele de calcul definite de relația (1.2), eroarea de trunchiere este cu atât mai mică cu cât numărul de termeni calculați este mai mare. Pe de altă parte, efortul de calcul pentru evaluarea valorii funcției creşte odată cu mărirea exactității impuse, dar şi cu creşterea modulului variabilei t.

Studiul comportamentului numeric al algoritmilor elaborați demonstrează că atunci când $|t| < \pi/4$, eroarea de trunchiere este mică ca valoare şi este comparabilă cu eroarea de rotunjire chiar şi la sumarea unui număr relativ mic de termeni ai seriei.

3.III. Modul de desfăşurare a lucrării

♦ **Cerințe față de îndeplinirea lucrării**

În baza pseudocodurilor descrise în secțiunea II a lucrării, să se elaboreze un program destinat calculului valorilor funcțiilor elementare din relația (1.1), utilizând dezvoltările în serii de puteri pentru acestea. În baza valorilor obținute, să se calculeze valoarea aproximativă a funcției $f(x)$, expresia analitică a căreia conține funcțiile elementare din relația (1.1). În plus, pentru fiecare dintre seriile date de relația (1.2) şi fiecare valoare x_i a variabilei independente x, programul va calcula şi va afişa suma parțială a seriei şi eroarea absolută de trunchiere pentru diferite ordine n. Se va reprezenta grafic variația erorii în funcție de numărul de termeni sumați. Se va analiza viteza de convergență a algoritmului pentru diferite valori x_i. Programul de calcul să se testeze pe exemplele propuse la sfârşitul lucrării de laborator.

Este necesar să se asigure (în caz de necesitate) afişarea valorilor funcției $f(x)$ într-un şir de puncte x_i, $i=1,N$, definite de către utilizator.

Să se extragă la tipar tabelul

i	x_i	y_i	ye_i	d_i	$k_{\max}$
1					
⋮					
N					

în care: i - numărul de rând; x_i - valoarea argumentului; y_i - valoarea funcției $f(x)$ în punctul x_i, calculată de program (utilizând algoritmul dezvoltării în serie de puteri); ye_i - valoarea funcției în punctul x_i, calculată cu ajutorul funcțiilor standard ale mediului de programare; $d_i = |y_i - ye_i|$; $k_{\max}$ - numărul maximal de termeni sumați din seriile folosite.

Programul trebuie să dea rezultat pentru orice valoare a argumentului x. Dacă pentru un careva x_i valoarea funcției $f(x)$ nu există sau are loc supraîncărcarea, atunci se va extrage la tipar comunicarea respectivă.

♦ **Chestiuni de studiat**

1. Precizați modalitățile prin care poate fi diminuată eroarea de calcul la sumarea numerică a unei serii.
2. Să se facă o analiză experimentală a timpului de calcul al valorilor funcțiilor elementare din relația (1.1), utilizând dezvoltarea în serie de puteri, în funcție de exactitatea de calcul ε impusă şi de mărimea argumentului x_0.
3. Să se reprezinte grafic variația erorii de calcul în funcție de numărul de termeni sumați ai seriei.

♦ **Probleme propuse spre rezolvare**

1. $f(x)=4\sin x - tgx + e^{-2x}$;	14. $f(x)=e^x\cos x + 2chxshx\sin x$;
2. $f(x)=1/\sin x + e^x\cos x - 4e^{-x}$;	15. $f(x)=e^{-2x}\cos x - 2chxshx\sin x$;
3. $f(x)=sh\,x + \cos x/(1-\cos x)$;	16. $f(x)=\cos^2 x + 4tgx - 12e^x$;
4. $f(x)=12e^x + \sin^2 x/(0.5-\cos x)$;	17. $f(x)=(\cos 2x + \sin 2x)/(e^{x^2}+1)$;
5. $f(x)=tgx + ctgx + e^x/(e^x+e^{-x})$;	18. $f(x)=e^{-2x}\sin 2x + chx\cos 2x$;

6. $f(x)=ctgx+(4e^{x}-e^{-x})/(2\sin x);$	19. $f(x)=e^{-3x}\sin 2x+shx\cos x;$
7. $f(x)=1/(tgx-ctgx)+2e^{x}-1;$	20. $f(x)=e^{x}\cos 4x+chx\cos 2x;$
8. $f(x)=(2\sin x-1)/(\cos x-e^{x});$	21. $f(x)=e^{-x}\sin x+chx-2\sin x/(e^{x}+e^{-2x});$
9. $f(x)=(2tgx-1)/(2-e^{-x});$	22. $f(x)=e^{2x}\cos x-3sh2x-2\sin x;$
10. $f(x)=4/\sin x+(e^{x}-\sin x)/\cos x;$	23. $f(x)=e^{-2x}\cos 2x+4ch2x-2chx\sin x;$
11. $f(x)=e^{x}tgx-\sin 2x/(0.4-\cos x);$	24. $f(x)=e^{-3x}/ch2x+2\cos 2x-4+2chx\sin x;$
12. $f(x)=\sin^{2}x+2ctgx+12e^{-x};$	25. $f(x)=e^{-2x}\cos x+4ch2x-3shx\sin 2x;$
13. $f(x)=2+e^{x}+(3+5\sin x)/(3\sin x-1);$	26. $f(x)=e^{-x}\cos 2x-ch2x-2shx\sin x.$

LUCRAREA DE LABORATOR №2
APROXIMAREA FUNCȚIILOR CU POLINOAME DE INTERPOLARE

2.I. Scopul lucrării

Studiul unor algoritmi de aproximare a funcțiilor continue cu polinomul algebric de interpolare şi implementarea acestora într-un limbaj de programare de nivel înalt.

2.II. Prezentarea teoretică a lucrării

♦ Formularea problemei

Fie pentru funcția continuă $f:[x_0;x_n]\to\mathbb{R}$ cunoscute valorile ei $y_i=f(x_i)$ în nodurile rețelei $\Delta:=\{x_i\}_{i=0}^n$, $a\le x_0<x_1<...<x_n\le b$ $(a,b\in\mathbb{R})$. Se cere de aproximat funcția $f(x)$ pe segmentul $[x_0;x_n]$ cu un polinom algebric de interpolare, adică de construit polinomul algebric $p_m(x)$ de grad minimal m, care în nodurile x_i, $i=\overline{0,n}$, ia valori egale cu y_i:

$$p_m(x_i)=y_i,\ i=\overline{0,n}. \tag{2.1}$$

Folosind reprezentarea analitică a polinomului obținut, să se calculeze valoarea aproximativă a funcției $f(x)$ în punctul $z\in[x_0;x_n]$, $z\ne x_i, i=\overline{0,n}$.

Problema de interpolare formulată are soluție unică [2-5; 8; 16; 20-21; 23-24]: pentru orice funcție continuă $f(x)$ şi orice sistem de noduri distincte $x_i, i=\overline{0,n}$, există un singur polinom algebric $p_m(x)$ de grad $m\le n$ ce satisface condițiile (2.1).

În funcție de necesitate, în practică sunt utilizate mai multe forme de reprezentare (algoritmi de construcție) ale polinomului de interpolare, unele dintre care le vom indica în continuare.

♦ Interpolare polinomială pe rețea neuniformă

Polinomul de interpolare sub forma lui Lagrange

Această formă a polinomului de interpolare este definită prin relația [3-5; 8; 23; 24, p.529]:

$$L_n(x)=\sum_{i=0}^{n} y_i l_i(x), \tag{2.2}$$

în care $l_i(x)=\prod_{j=0,\,j\ne i}^{n}\left((x-x_j)/(x_i-x_j)\right)$ sunt numite polinoamele fundamentale de interpolare Lagrange.

Pseudocodul algoritmului

Date de intrare: z (de tip real) – punctul în care se calculează valoarea polinomului de interpolare; n (de tip întreg) – gradul maximal al polinomului de interpolare; $x[n]$ - tabloul (cu elemente de tip real; indici de la 0 la n) nodurilor de interpolare; $y[n]$ - tabloul (cu elemente de tip real; indici de la 0 la n) valorilor funcției $f(x)$ în nodurile de interpolare. Date de ieşire: z (de tip real) – punctul în care se calculează valoarea polinomului de interpolare; p (de tip real) – valoarea polinomului de interpolare în punctul z; întreg i, j real t, p
$p:=0$ pentru $i:=\overline{0,n}$ execută { $t:=1$

```
        pentru j := 0,n execută { dacă j ≠ i atunci t := t·(z − x_j)/(x_i − x_j) }
        p := p + t y_i
    }
scrie z, p
```

Analiza algoritmului

În cazul în care $f(x)$ este continuă pe $[x_0; x_n]$ şi mai posedă şi derivate continui până la ordinul $n+1$ inclusiv (vom utiliza notația $f(x) \in C^{n+1}[x_0; x_n]$), este adevărată următoarea estimație pentru eroarea $E_n^L(x) := f(x) - L_n(x)$ a procesului de interpolare în punctul $z \in [x_0; x_n]$ [2-3; 8; 10; 16; 23; 24, p.537]:

$$\left|E_n^L(z)\right| \le \left|\omega_n(z)\right| M_{n+1}/(n+1)!,$$

unde $M_{n+1} = \max\limits_{x\in[x_0;x_n]} \left|f^{(n+1)}(x)\right|$, $\omega_n(z) = (z-x_0)(z-x_1)\cdot\ldots\cdot(z-x_n)$. Analiza comportamentului funcției $\omega_n(x)$ permite să concluzionăm că eroarea interpolării $E_n^L(z)$ va fi în medie mai mare dacă punctul z se află în vecinătatea extremităților segmentului $[x_0; x_n]$ [21, p.63].

Efortul de calcul pentru evaluarea polinomului Lagrange în punctul z conform formulei (2.2), este de $4n^2 + 6n + 2$ operații aritmetice (aditive şi multiplicative). Complexitatea de ordinul $O(4n^2)$ uneori este inacceptabilă în practică şi poate să conducă la apariția unor erori de calcul mari. Din această cauză au fost elaborați alți algoritmi de calcul al valorilor polinomului de interpolare.

Dacă se calculează valorile polinomului Lagrange într-un număr mare de puncte (diferite de nodurile rețelei de interpolare), atunci este avantajoasă următoarea formă:

$$L_n(x) = \prod_{j=0}^{n}(x - x_j) \cdot \sum_{i=0}^{n}\left(r_i/(x - x_i)\right), \qquad (2.3)$$

în care coeficienții r_i $(i = \overline{0,n})$ sunt dați de relația $r_i = y_i \Big/ \prod\limits_{j=0,\, j\ne i}^{n}(x_i - x_j)$. Procesul de calcul descris de formula (2.3) admite următorul pseudocod:

```
Date de intrare:
   z  (de tip real) – punctul în care se calculează valoarea polinomului de interpolare;
   n  (de tip întreg) – gradul maximal al polinomului de interpolare;
   x[n] - tabloul (cu elemente de tip real; indici de la 0 la n) nodurilor de interpolare;
   y[n] - tabloul (cu elemente de tip real; indici de la 0 la n) valorilor funcției f(x) în nodurile de interpolare.
Date de ieşire:
   z  (de tip real) – punctul în care se calculează valoarea polinomului de interpolare;
   p  (de tip real) – valoarea polinomului de interpolare în punctul z;
întreg i, j
real t, p, r
```

```
t := 1; p := 0
pentru j := 0,n execută t := t(z − x_j)
pentru i := 0,n execută
 {
    r := y_i
    pentru j := 0,n execută { dacă j ≠ i atunci r := r/(x_i − x_j) }
    p := p + r/(z − x_i)
 }
p := p t
```

scrie z, p

Spre deosebire de cazul în care este utilizată formula (2.2), calculul valorii polinomului $L_n(x)$, conform relației (2.3), necesită $2n^2+7n+6$ operații aritmetice. Dacă se efectuează m evaluări, efortul global de calcul este de $2n^2+(5m+2)n+6m$ operații. Din punctul de vedere al eficienței spațiale, variantele (2.2) şi (2.3) necesită fiecare un spațiu de memorie de ordin $O(2n)$.

Algoritmul lui Neville-Aitken

Calculul valorii aproximative a funcției într-un punct dat poate fi efectuat şi fără a apela la forma analitică a polinomului Lagrange. Aceasta se poate efectua, de exemplu, utilizând algoritmul lui Neville-Aitken [2, p.12; 25, p.116]. Procedura iterativă de calcul utilizează pentru început două noduri de interpolare, ca apoi treptat să includă în calcule noi noduri până se obține precizia dorită.

Fie $P_{j,j+1,\dots,j+i}(x)$ polinomul de interpolare construit pe nodurile distincte $x_j, x_{j+1}, \dots, x_{j+i}$. Evidențiem cazul $i=0$ când vom considera că $P_j(x)=y_j$, $j=\overline{0,n}$. Polinoamele $P_{j,j+1,\dots,j+i}(x)$ satisfac relația de recurență [2, p.12]:

$$P_{j,j+1,\dots,j+i}(x)=\left((x_{j+i}-x)P_{j,j+1,\dots,j+i-1}(x)+(x-x_j)P_{j+1,j+2,\dots,j+i}(x)\right)\Big/(x_{j+i}-x_j). \qquad (2.4)$$

Algoritmul lui Neville-Aitken presupune calculul consecutiv conform formulei (2.4) a elementelor evidențiate cu bold din tabelul de mai jos:

x_0	$y_0 = P_0$					
		$\mathbf{P_{01}}$				
x_1	$y_1 = P_1$		$\mathbf{P_{012}}$			
		$\mathbf{P_{12}}$		$\mathbf{P_{0123}}$		
x_2	$y_2 = P_2$		$\mathbf{P_{123}}$		$\ddots$	
		$\mathbf{P_{23}}$				$\mathbf{P_{0,\dots,n-1}}$
x_3	$y_3 = P_3$					
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$		$\mathbf{P_{0,1,\dots,n}}$
x_{n-3}	$y_{n-3} = P_{n-3}$					
		$\mathbf{P_{n-3,n-2}}$				$\mathbf{P_{1,\dots,n}}$
x_{n-2}	$y_{n-2} = P_{n-2}$		$\mathbf{P_{n-3,n-2,n-1}}$		$\iddots$	
		$\mathbf{P_{n-2,n-1}}$		$\mathbf{P_{n-3,n-2,n-1,n}}$		
x_{n-1}	$y_{n-1} = P_{n-1}$		$\mathbf{P_{n-2,n-1,n}}$			
		$\mathbf{P_{n-1,n}}$				
x_n	$y_n = P_n$					

Coloanele evidențiate ale tabelului se completează de la stânga spre dreapta. Fiecare element al coloanei se calculează recursiv în baza celor doi "vecini" din coloana precedentă. Algoritmul admite următoarea reprezentare în pseudocod:

Date de intrare:
- z (de tip real) – punctul în care se calculează valoarea polinomului de interpolare;
- n (de tip întreg) – gradul maximal al polinomului de interpolare;
- $x[n]$ - tabloul (cu elemente de tip real; indici de la 0 la n) nodurilor de interpolare;
- $y[n]$ - tabloul (cu elemente de tip real; indici de la 0 la n) valorilor funcției $f(x)$ în nodurile de interpolare.

Date de ieşire:
- z (de tip real) – punctul în care se calculează valoarea polinomului de interpolare;
- y_n (de tip real) – valoarea polinomului de interpolare în punctul z;

întreg i, j

pentru $i := \overline{1,n}$ execută
{ pentru $j := \overline{n,i}$ execută $y_j := \left((x_j - x)y_{j-1} + (z - x_{j-i})y_j\right) / (x_j - x_{j-i})$ }
scrie z, y_n

Schema descrisă permite să evidențiem un algoritm de calcul al valorii funcției $f(x)$ într-un punct dat $z \neq x_i$ $(i = \overline{0,n})$ cu exactitatea dorită $\varepsilon > 0$ sau cu cea mai bună exactitate posibilă pentru informația avută. Vom nota $L_m(x) := P_{0,1,\dots,m}(x)$, $m = \overline{0,n}$. Pentru a descrie algoritmul menționat vom utiliza relația $E_m^L(x) \approx L_{m+1}(x) - L_m(x)$ [20, p.48]. Astfel, mărimea $\varepsilon_m := \left|P_{0,1,\dots,m+1}(x) - P_{0,1,\dots,m}(x)\right| \equiv \left|L_{m+1}(x) - L_m(x)\right|$ poate fi considerată ca o aproximație a erorii interpolării $E_m^L(x)$.

Se calculează succesiv valorile: $L_0(x) := P_0(x)$, $L_1(x) := P_{0,1}(x)$, $\varepsilon_0 := \left|L_1(x) - L_0(x)\right|$, $L_2(x) := P_{0,1,2}(x)$, $\varepsilon_1 := \left|L_2(x) - L_1(x)\right|$, $L_3(x) := P_{0,1,2,3}(x)$, $\varepsilon_2 := \left|L_3(x) - L_2(x)\right|, \dots$. Dacă pentru o valoare a numărului m $(0 \le m < n)$ va avea loc inegalitatea $\varepsilon_m < \varepsilon$, atunci calculele se stopează şi se consideră $f(x) \approx L_{m+1}(x)$. În caz contrar, $(\varepsilon_m \ge \varepsilon, \forall m = \overline{0, n-1})$ se determină cel mai mic m pentru care $\varepsilon_m = \min\limits_{0 \le j \le n-1} \varepsilon_j$ şi se consideră $f(x) \approx L_{m+1}(x)$.

Pseudocodul algoritmului

Date de intrare:
z (de tip real) – punctul în care se calculează valoarea polinomului de interpolare;
n (de tip întreg) – gradul maximal al polinomului de interpolare;
$x[n]$ - tabloul (cu elemente de tip real; indici de la 0 la n) nodurilor de interpolare;
$y[n]$ - tabloul (cu elemente de tip real; indici de la 0 la n) valorilor funcției $f(x)$ în nodurile de interpolare;
ε (de tip real) – exactitatea de calcul dorită.
Date de ieşire:
z (de tip real) – punctul în care se calculează valoarea polinomului de interpolare;
p (de tip real) – valoarea polinomului de interpolare în punctul z;
er (de tip real) – cea mai bună exactitate posibilă pentru informația avută;
întreg i, j, k
real d, t, p, er

$er := -1$; $t := y_0$
pentru $i := \overline{1,n}$ execută
{
pentru $k := \overline{i,1}$ execută { $j := k - 1$; $y_j := y_k + (y_k - y_j)(z - x_i) / (x_i - x_j)$ }
$d := \left|y_0 - t\right|$; $t := y_0$
dacă $d < \varepsilon$ atunci { $er := \varepsilon$; $p := y_0$; se iese din ciclu }
altfel { dacă ($er < 0$ sau $d < er$) atunci { $er := d$; $p := y_0$ } }
}
scrie z, p, er

Analiza algoritmului

Efortul de calcul necesar pentru realizarea algoritmului este de cel mult $3.5n^2 + 4.5n$ operații aritmetice, iar din punct de vedere al eficienței spațiale necesită un spațiu de memorie de ordin $O(2n)$.

Polinomul de interpolare sub forma lui Newton cu diferențe divizate

Această formă a polinomului de interpolare are următorul aspect:

$$N_n(x) = y_0 + \sum_{i=1}^{n} \delta f[x_0; \dots; x_i] \prod_{j=0}^{i-1} (x - x_j). \qquad (2.5)$$

Relația (2.5) utilizează diferențele divizate de ordinul k ale funcției $f(x)$:

$$\delta f[x_i;...;x_{i+k}] := \left(\delta f[x_{i+1};...;x_{i+k}] - \delta f[x_i;...;x_{i+k-1}]\right) / \left(x_{i+k} - x_i\right) \ (i,k \in \mathbb{N}). \qquad (2.6)$$

În particular, pentru $i = 0, k = \overline{1,n}$ obținem diferențele divizate din formula (2.5). Vom considera că $\delta f[x_j] = y_j,\ j = \overline{0,n}$.

Evaluarea valorii polinomului (2.5) se va efectua în modul următor:

1. Utilizând definiția (2.6) se generează diferențele divizate. Vom ține cont că în procesul de construcție a polinomului participă efectiv doar diferențele divizate $\delta f[x_0;...;x_i],\ i = \overline{1,n}$, de aceea restul diferențelor le vom considera ca date intermediare. Pentru a economisi memoria, vom amplasa diferențele divizate în tabloul în care inițial se păstrau valorile funcției $f(x)$ în noduri. Tabloul dat se va înnoi parțial la calcularea diferențelor de ordin superior. Completarea tabloului se va face începând cu ultimul element al acestuia. Astfel, la calcularea diferențelor divizate de ordinul I, elementul y_n se va înlocui cu $\delta f[x_{n-1};x_n]$, y_{n-1} - cu $\delta f[x_{n-2};x_{n-1}]$ ș.a.m.d. Elementul y_0 rămâne nemodificat. La calcularea diferențelor de ordinul II, primele două elemente ale tabloului rămân nemodificate, iar restul elementelor sunt înlocuite cu diferențele respective. În momentul în care a fost calculată diferența divizată de ordinul n, în locul vectorului inițial $(y_0;y_1;...;y_n)^T$ avem vectorul $(y_0;\delta f[x_0;x_1];\delta f[x_0;x_1;x_2];...\ ...;\delta f[x_0;x_1;...;x_n])^T$.

2. Din motive de eficiență, evaluarea polinomului de interpolare Newton în punctul z se realizează conform algoritmului (schemei) lui Horner [24, p.74; 25, p.24]. Pentru aceasta polinomul $N_n(x)$ se scrie sub forma:

$$N_n(x) = y_0 + (x-x_0)\Big(\delta f[x_0;x_1] + (x-x_1)\Big(\delta f[x_0;x_1;x_2] + ... + (x-x_{n-2})\Big(\delta f[x_0;x_1;...;x_{n-1}] + (x-x_{n-1})\underbrace{\delta f[x_0;x_1;...;x_n])...}_{n-1}\Big)\Big).$$

Vom afla valoarea lui $N_n(x)$ în punctul z, calculând succesiv mărimile:

$$b_n := \delta f[x_0;x_1;...;x_n],\ b_k := \delta f[x_0;x_1;...;x_k] + (z-x_k)b_{k+1},\ k = \overline{n-1,1},\ b_0 = N_n(x) := y_0 + (z-x_0)b_1.$$

Pseudocodul algoritmului

Date de intrare:
- z (de tip real) – punctul în care se calculează valoarea polinomului de interpolare;
- n (de tip întreg) – gradul maximal al polinomului de interpolare;
- $x[n]$ - tabloul (cu elemente de tip real; indici de la 0 la n) nodurilor de interpolare;
- $y[n]$ - tabloul (cu elemente de tip real; indici de la 0 la n) valorilor funcției $f(x)$ în nodurile de interpolare.

Date de ieşire:
- z (de tip real) – punctul în care se calculează valoarea polinomului de interpolare;
- p (de tip real) – valoarea polinomului de interpolare în punctul z;

întreg i,k
real p

pentru $i := \overline{1,n}$ execută { pentru $k := \overline{n,i}$ execută $y_k := (y_k - y_{k-1})/(x_k - x_{k-i})$ }
$p := y_n$
pentru $i := \overline{n-1,0}$ execută $p := y_i + (z - x_i)p$
scrie z, p

Analiza algoritmului

În cazul în care sunt cunoscute diferențele divizate, calculul direct al valorii polinomului $N_n(x)$ conform formulei (2.5) (în sensul că se găsesc valorile tuturor termenilor, apoi aceştia se sumează),

necesită n^2+2n operații aritmetice. În schimb, schema lui Horner permite obținerea valorii lui $N_n(x)$, efectuând doar $3n$ operații. În plus, utilizarea schemei menționate nu numai că economiseşte timpul de calcul, dar şi îmbunătățește exactitatea calculelor în baza diminuării erorilor de rotunjire (numărul mare de operații poate duce la pierderea exactității datorită acumulării erorilor de rotunjire). Realizarea celor două etape ale algoritmului necesită $1.5n^2+4.5n$ operații aritmetice cu numere reale.

În cazul în care este necesar să se calculeze valorile polinomului de interpolare $N_n(x)$ în mai multe puncte, diferențele divizate $\delta f[x_0;...;x_i]\ (i=\overline{1,n})$ se calculează doar o singură dată, iar pentru a calcula valorile cerute se apelează doar la algoritmul lui Horner. Dacă se efectuează m evaluări, efortul global de calcul este de $1.5(n^2+n)+3mn$ operații cu numere reale. Spațiul de memorie necesar pentru realizarea algoritmului este de ordinul $O(2n)$. Timpul de calcul este dependent de eroarea impusă, având valori mari doar în cazurile în care se doreşte o precizie ridicată.

Algoritmul permite şi mărirea gradului polinomului de interpolare cu un efort minim de calcul, întrucât la adăugarea unui nou nod în rețeaua de interpolare se reutilizează coeficienții de la gradul anterior.

În cazul în care $f(x)$ este continuă, se poate utiliza următoarea formulă pentru eroarea $E_n^N(x):=f(x)-N_n(x)$ a procesului de interpolare în punctul $z\in[x_0;x_n]$:

$$E_n^N(z)=\delta f[x_0;x_1;...;x_n;z]\omega_n(z),\ \omega_n(z)=(z-x_0)\cdot...\cdot(z-x_n).$$

Remarcă. În cazul interpolării pe o rețea neuniformă, se recomandă a alege în calitate de noduri următoarele puncte [24, p.541]:

$$x_i=(b+a)/2+t_i\cdot(b-a)/2,\ \ i=\overline{0,n},$$

unde $t_i=-\cos((2i+1)\pi/(2n+2))\ (i=\overline{0,n})$ sunt zerourile polinomului Chebyshev de speța I de gradul $n+1$; $[a;b](\supseteq[x_0;x_n])$ - segment pe care este definită funcția $f(x)$. Prin aceasta se obține, în primul rând, micşorarea influenței erorii de calcul asupra rezultatului, iar în al doilea rând, în cazul unei funcții suficient de netede, eroarea se micşorează odată cu creşterea lui n. Evaluarea erorii de interpolare pentru cazul în care $f(x)\in C^{n+1}[x_0;x_n]$ este dată de formula:

$$|E_n(z)|\le\frac{M_{n+1}}{(n+1)!}\cdot\frac{(x_n-x_0)^{n+1}}{2^{2n+1}};\ M_{n+1}=\max_{x\in[x_0;x_n]}|f^{(n+1)}(x)|.$$

- **Interpolare polinomială pe rețea uniformă**

Polinomul de interpolare sub forma lui Lagrange cu noduri echidistante

În cazul în care nodurile rețelei Δ sunt echidistante, adică are loc relația

$$x_i=x_0+ih,\ i=\overline{0,n},\ h>0,$$

formula polinomului de interpolare Lagrange (2.2) se simplifică şi capătă următorul aspect:

$$L_n(x)\equiv L_n(x_0+th)=\frac{(-1)^n}{n!}\left(\prod_{j=0}^{n}(t-j)\right)\sum_{i=0}^{n}(-1)^i C_n^i\frac{y_i}{t-i},$$

unde $t=(x-x_0)/h$, iar $C_n^i=n!/(i!(n-i)!)$ - numărul combinărilor din n elemente luate câte i. Se vor

utiliza două procedee de optimizare a algoritmului de calcul: primul - bazat pe formula de recurență $C_k^i = C_{k-1}^{i-1} + C_{k-1}^i$ $(k = \overline{2,n},\ i = \overline{1,k-1};\ C_{k-1}^0 = 1)$ (triunghiul lui Pascal), iar al doilea – pe formula combinărilor complementare $C_n^i = C_n^{n-i}$ $(i = \overline{0,\lfloor n/2 \rfloor},\ n \in \mathbb{N},\ \lfloor n/2 \rfloor$ - partea întreagă a numărului $n/2)$.

Pseudocodul algoritmului

Date de intrare:
z (de tip real) – punctul în care se calculează valoarea polinomului de interpolare;
n (de tip întreg) – gradul maximal al polinomului de interpolare;
x_0, x_n - (de tip real) primul şi respectiv, ultimul nod al rețelei de interpolare;
$y[n]$ - tabloul (cu elemente de tip real; indici de la 0 la n) valorilor funcției $f(x)$ în nodurile de interpolare.
Date de ieşire:
z (de tip real) – punctul în care se calculează valoarea polinomului de interpolare;
p (de tip real) – valoarea polinomului de interpolare în punctul z;
întreg i, j, k
real h, t, p, s
tablou real $c[k]$ (tabloul (cu elemente de tip real; indici de la 0 la k) ce conține numărul combinărilor din n elemente luate câte i, $i = \overline{0,k}$)

$k := \lfloor n/2 \rfloor;\ h := (x_n - x_0)/n;\ t := (z - x_0)/h;\ p := (-1)^n t$
pentru $j := \overline{1,n}$ execută $p := p(t-j)/j$
$c_0 := 1$
pentru $i := \overline{1,k}$ execută $c_i := 0$
pentru $j := \overline{1,k}$ execută { pentru $i := \overline{j,1}$ execută $c_i := c_{i-1} + c_i$ }
pentru $j := \overline{k+1,n}$ execută { pentru $i := \overline{k,1}$ execută $c_i := c_{i-1} + c_i$ }
$s := 0;\ j := n \bmod 2$
dacă $j = 1$ atunci { pentru $i := \overline{0,k}$ execută $s := s + (-1)^i c_i (y_i/(t-i) - y_{n-i}/(t-(n-i)))$ }
altfel
{
pentru $i := \overline{0,k-1}$ execută $s := s + (-1)^i c_i (y_i/(t-i) + y_{n-i}/(t-(n-i)))$
$s := s + (-1)^k c_k\ y_k/(t-k)$
}
$p := p\ s$
scrie z, p

Analiza algoritmului

Realizarea algoritmului necesită un efort de calcul de $O(0.375n^2 + 3.75n)$ operații aritmetice şi un spațiu de memorie de $O(1.5n)$.

Polinomul de interpolare sub forma lui Newton cu noduri echidistante

A) Formula lui Newton de interpolare ascendentă are aspectul următor:

$$N_n(x) \equiv N_n(x_0 + th) = y_0 + \sum_{k=1}^{n} \frac{\Delta^k y_0}{k!} \prod_{j=0}^{k-1} (t-j), \tag{2.7}$$

unde $t = (x - x_0)/h,\ h > 0$ - pasul tabelei de valori. Relația (2.7) utilizează diferențele finite de ordinul k ale funcției $f(x)$:

$$\Delta^k y_i = \Delta^{k-1} y_{i+1} - \Delta^{k-1} y_i,\ k = \overline{1,n},\ i = \overline{0,n-k}. \tag{2.8}$$

În particular, pentru $i = 0,\ k = \overline{1,n}$, obținem diferențele finite din formula (2.7). Vom considera că

$\Delta^0 y_i = y_i = f(x_i),\ x_i = x_0 + ih,\ i = \overline{0,n}$.

Evaluarea valorii polinomului (2.7) se va efectua în modul următor:

1. Utilizând definiția (2.8), se generează diferențele finite. Vom ține cont că în procesul de construcție a polinomului participă efectiv doar diferențele finite $\Delta^k y_0,\ k = 1,...,n$, de aceea restul diferențelor le vom considera ca date intermediare. Pentru a economisi memoria, vom amplasa diferențele în tabloul în care inițial se păstrau valorile funcției $f(x)$ în noduri. Tabloul dat se va înnoi parțial la calcularea diferențelor de ordin superior. Completarea tabloului va începe cu ultimul element al acestuia. Astfel, la calcularea diferențelor finite de ordinul I, elementul y_n se va înlocui cu Δy_{n-1}, y_{n-1} - cu Δy_{n-2} ş.a.m.d. Elementul y_0 nu se modifică. La calcularea diferențelor de ordinul II, primele două elemente ale masivului nu suferă modificații, iar restul elementelor sunt înlocuite cu diferențele respective. În momentul în care a fost calculată diferența finită de ordinul n, în locul vectorului inițial $(y_0; y_1; ...; y_n)^T$ obținem vectorul $(y_0; \Delta y_0; \Delta^2 y_0; ...; \Delta^n y_0)^T$.

2. Evaluarea polinomului (2.7) în punctul z se efectuează conform algoritmului lui Horner. Pentru aceasta $N_n(x_0 + th)$ se scrie sub forma:

$$N_n(x_0 + th) = y_0 + t\left(\Delta y_0 + (t-1)\left(\frac{\Delta^2 y_0}{2!} + ... + (t-n+2)\left(\frac{\Delta^{n-1} y_0}{(n-1)!} + (t-n+1)\frac{\Delta^n y_0}{n!}\right)...\right)\right).$$

Vom afla valoarea acestui polinom în punctul z, calculând succesiv mărimile:

$$t = (z - x_0)/h,\ b_n := \Delta^n y_0 / n!,\ b_k := \frac{\Delta^k y_0}{k!} + (t-k)b_{k+1},\ k = \overline{n-1,1},\ b_0 = N_n(x_0 + th) := y_0 + t b_1.$$

Pseudocodul algoritmului

Date de intrare:
- z (de tip real) – punctul în care se calculează valoarea polinomului de interpolare;
- n (de tip întreg) – gradul maximal al polinomului de interpolare;
- x_0, x_n - (de tip real) primul şi respectiv, ultimul nod al rețelei de interpolare;
- $y[n]$ - tabloul (cu elemente de tip real; indici de la 0 la n) valorilor funcției $f(x)$ în nodurile de interpolare.

Date de ieşire:
- z (de tip real) – punctul în care se calculează valoarea polinomului de interpolare;
- p (de tip real) – valoarea polinomului de interpolare în punctul z;

întreg i, k, u
real h, t, p

pentru $i := \overline{1,n}$ execută { pentru $k := \overline{n,i}$ execută $y_k := y_k - y_{k-1}$ }
$u := 1$
pentru $i := \overline{2,n}$ execută { $u := ui$; $y_i := y_i / u$ }
$h := (x_n - x_0)/n$; $t := (z - x_0)/h$; $p := y_n$
pentru $i := \overline{n-1,0}$ execută $p := y_i + (t-i)p$
scrie z, p

Analiza algoritmului

În cazul în care sunt cunoscute diferențele finite, pentru realizarea algoritmului este necesar un efort de calcul de $0.5n^2 + 6.5n - 3$ operații aritmetice cu numere reale şi un spațiu de memorie de $O(n)$.

În cazul în care $f(x) \in C^{n+1}[x_0; x_n]$ este adevărată următoarea estimație pentru eroarea procesului de

interpolare în punctul $z \in [x_0; x_n]$ [8; 16; 24, p.538]:

$$\left|E_n^N(z)\right| \le \left|t(t-1)\cdot\ldots\cdot(t-n)\right| h^{n+1} M_{n+1} / (n+1)!,$$

unde $M_{n+1} = \max\limits_{x\in[x_0;x_n]} \left| f^{(n+1)}(x)\right|$, $t = (z - x_0)/h$.

B) Formula lui Newton de interpolare descendentă are aspectul următor:

$$N_n(x) \equiv N_n(x_n + th) = y_n + \sum_{k=1}^{n} \frac{\Delta^k y_{n-k}}{k!} \prod_{j=0}^{k-1} (t + j), \tag{2.9}$$

unde $\Delta^k y_{n-k}$ sunt diferențe finite de ordinul k, $k = \overline{1,n}$, $t = (x - x_n)/h$ (deoarece $x \in (x_0; x_n)$ avem $t < 0$), $x_i = x_0 + ih$, $i = \overline{0,n}$, $h > 0$ - pasul tabelului de valori.

Ca şi în cazul precedent **A)**, se vor calcula diferențele finite în baza relației (2.8) şi se vor memora în tabloul valorilor funcției $f(x)$ în noduri. Vom menționa, totuşi, că în cazul dat tabloul se va completa începând cu primul element: elementul y_0 se va înlocui cu Δy_0, y_1 - cu Δy_1 ş.a.m.d. În momentul în care a fost calculată diferența finită de ordinul n, în locul vectorului inițial $(y_0; y_1; \ldots; y_n)^T$ obținem $(\Delta^n y_0; \Delta^{n-1} y_1; \ldots; \Delta y_{n-1}; y_n)^T$. Evaluarea polinomului (2.9) în punctul z se efectuează în baza algoritmului lui Horner:

$$t = (z - x_n)/h,\ b_n := \frac{\Delta^n y_0}{n!},\ b_k := \frac{\Delta^k y_{n-k}}{k!} + (t + k) b_{k+1},\ k = \overline{n-1,1},\ b_0 = N_n(x_n + th) := y_n + t b_1.$$

Pseudocodul algoritmului

Date de intrare:
- z (de tip real) – punctul în care se calculează valoarea polinomului de interpolare;
- n (de tip întreg) – gradul maximal al polinomului de interpolare;
- x_0, x_n - primul şi, respectiv, ultimul nod (de tip real) al rețelei de interpolare;
- $y[n]$ - tabloul (cu elemente de tip real; indici de la 0 la n) valorilor funcției $f(x)$ în nodurile de interpolare.

Date de ieşire:
- z (de tip real) – punctul în care se calculează valoarea polinomului de interpolare;
- p (de tip real) – valoarea polinomului de interpolare în punctul z;

întreg i, k, u
real h, t, p

```
pentru i := 1,n execută { pentru k := 0,n-i execută  y_k := y_{k+1} - y_k }
u := 1
pentru i := 2,n execută { u := u i; y_{n-i} := y_{n-i} / u }
h := (x_n - x_0)/n; t := (z - x_n)/h; p := y_0
pentru i := n-1,0 execută  p := y_{n-i} + (t + i) p
scrie z, p
```

Analiza algoritmului

În cazul în care sunt cunoscute diferențele finite, pentru realizarea algoritmului este necesar un efort de calcul de $0.5n^2 + 6.5n - 3$ operații aritmetice cu numere reale şi un spațiu de memorie de $O(n)$.

În cazul în care $f(x) \in C^{n+1}[x_0; x_n]$ este adevărată următoarea estimație pentru eroarea procesului de interpolare în punctul $z \in [x_0; x_n]$ [24, p.538]:

$$\left|E_n^N(z)\right| \le \left|t(t+1)\cdot\ldots\cdot(t+n)\right| h^{n+1} M_{n+1} / (n+1)!,$$

unde $M_{n+1} = \max\limits_{x\in[x_0;x_n]} \left| f^{(n+1)}(x) \right|$, $t = (z - x_n)/h$.

Remarcă. Din punctul de vedere al exactității calculelor este rezonabil să se utilizeze formula Newton de interpolare ascendentă pentru calculul valorii funcției în punctele jumătății din stânga a segmentului $[x_0; x_n]$, iar formula Newton de interpolare descendentă – în punctele jumătății din dreapta a segmentului $[x_0; x_n]$.

3.III. Modul de desfăşurare a lucrării

♦ **Cerințe față de îndeplinirea lucrării**

Fie dată analitic funcția $y = f(x), x \in [a;b]$. În baza pseudocodurilor descrise în secțiunea II a lucrării, să se elaboreze un program, care are drept obiectiv realizarea următoarelor etape:

1) Calculul valorilor $y_i = f(x_i), i = \overline{0,n}$ $(n \in \mathbb{N})$ (cu ajutorul funcțiilor standard ale mediului de programare) în punctele distincte x_i, $a \le x_0 < x_1 < ... < x_n \le b$.
2) Calculul valorii aproximative a funcției $f(x)$ în punctul $x = z$, utilizând polinomul de interpolare sub forma lui Lagrange sau Newton.
3) Calculul aproximativ cu exactitatea $\varepsilon = 10^{-6}$ (sau cu cea mai bună exactitate posibilă) al valorii funcției $f(x)$ în punctul $x = z$, utilizând algoritmul lui Neville-Aitken.

Să se analizeze variația erorii la aproximarea prin interpolare a funcției $f(x)$ în $x = z$ (a se vedea punctul 2)) în funcție de:

2.1. Numărul nodurilor de interpolare;

2.2. Distribuția nodurilor de interpolare:

a) noduri echidistante;
b) nodurile sunt în concordanță cu zerourile polinomului Chebyshev de gradul $n+1$ definit pe $[a;b] \supseteq [x_0; x_n]$;
c) alte modalități de amplasare;

2.3. Poziția punctului z (în vecinătatea extremității din stânga sau din dreapta a segmentului de interpolare, sau în vecinătatea mijlocului acestuia).

Să se testeze programul de calcul pe exemplele propuse la sfârşitul lucrării de laborator.

Să se extragă la tipar tabelele (pentru fiecare polinom de interpolare):

Tabelul după care se interpolează (suportul problemei de interpolare)

x_i	x_0	...	x_n
y_i	y_0	...	y_n

x_i $(i = \overline{0,n})$ - nodurile rețelei de interpolare;

y_i $(i = \overline{0,n})$ - valorile funcției $f(x)$ în noduri;

Tabelul interpolării şi testării

j	z_j	P_j	f_j	Δ_j
1				
⋮				
m				

z_j $(j = \overline{1,m})$ - punctele în care se interpolează valoarea funcției $f(x)$;

P_j $(j = \overline{1,m})$ - valorile polinomului de interpolare în punctele z_j;

f_j $(j = \overline{1,m})$ - valorile funcției $f(x)$ în punctele z_j, calculate cu ajutorul funcțiilor standard ale mediului de programare;

Δ_j ($j=\overline{1,m}$) - erorile de interpolare $\Delta_j = \left| f_j - P_j \right|$ în punctele z_j.

Programul trebuie să dea rezultat pentru orice sistem finit de puncte de interpolare.

♦ **Chestiuni de studiat**

1. Precizați modalitățile prin care poate fi diminuată eroarea interpolării.
2. Să se facă o analiză experimentală a timpului de calcul al valorilor polinomului de interpolare în funcție de gradul polinomului.
3. Să se formuleze exemple de funcții continue pentru care polinoamele de interpolare corespunzătoare nu vor converge uniform.
4. Să se construiască polinoamele de interpolare Lagrange şi Newton ce aproximează funcția $f(x)=1/(1+25x^2)$, $x\in[-1;1]$. Să se reprezinte grafic funcția $f(x)$ şi polinomul de interpolare corespunzător, precum şi variația erorii în funcție de numărul nodurilor de interpolare dacă:
   - nodurile rețelei de pe $[-1;1]$ sunt echidistante;
   - nodurile rețelei sunt zerourile polinomului Chebyshev de gradul $n+1$.
5. Analizați comportarea aproximării funcției prin interpolare polinomială în exteriorul segmentului $[x_0; x_n]$ (problema extrapolării).

♦ **Probleme propuse spre rezolvare**

1. $f(x)=e^{x+2}\sin x/(x+1)$, $x\in[0;2]$;
2. $f(x)=e^x\sqrt{x^2+1}$, $x\in[0.5;4]$;
3. $f(x)=e^x\sin x/x$, $x\in[1;4]$;
4. $f(x)=x/\ln x$, $x\in[2;5]$;
5. $f(x)=e^x\cos x/x$, $x\in[2;4]$;
6. $f(x)=\ln(x+1)/x$, $x\in[10;15]$;
7. $f(x)=e^x\ln x$, $x\in[2;4]$;
8. $f(x)=xe^{\sin x}$, $x\in[4;7]$;
9. $f(x)=e^{-x}/x$, $x\in[0.2;2]$;
10. $f(x)=e^{-x}/x^2$, $x\in[1.1;4]$;
11. $f(x)=xe^{2\sin x}$, $x\in[1;4]$;
12. $f(x)=xe^{\cos x}$, $x\in[10;16]$;
13. $f(x)=3\sin x+\cos 2x$, $x\in[-\pi/2;0]$;
14. $f(x)=(x-1)^2-0.5e^x$, $x\in[0;3]$;
15. $f(x)=(x-1)^2-e^{-x}$, $x\in[0;2.5]$;
16. $f(x)=x^3-\sin x$, $x\in[0.6;3]$;
17. $f(x)=4x-\cos x$, $x\in[0;5]$;
18. $f(x)=x^2-\sin x$, $x\in[-1;4]$;
19. $f(x)=\sin x+\cos x$, $x\in[-\pi/3;\pi/3]$;
20. $f(x)=x^2-\cos x$, $x\in[-2;4]$;
21. $f(x)=\cos x-\sin x$, $x\in[-\pi/2;\pi/2]$;
22. $f(x)=x-2\cos 0.5x$, $x\in[-2;3]$;
23. $f(x)=2x-\cos x$, $x\in[-2;3]$;
24. $f(x)=x^2+\ln(x+5)$, $x\in[-4;3]$;
25. $f(x)=\sin 3x-4\cos 3x$, $x\in[0;\pi/2]$;
26. $f(x)=x^2+\lg x$, $x\in[2;6]$.

LUCRAREA DE LABORATOR №3
APLICAREA FORMULELOR DE CUADRATURĂ LA CALCULUL APROXIMATIV AL INTEGRALEI

3.I. Scopul lucrării

Studiul unor algoritmi de integrare numerică şi implementarea acestora într-un limbaj de programare de nivel înalt.

3.II. Prezentarea teoretică a lucrării

♦ Formularea problemei

Fie $f:[a;b]\to\mathbb{R}$ $(-\infty<a<b<\infty)$ o funcție continuă. Să se calculeze integrala definită

$$I=\int_a^b f(x)dx, \tag{3.1}$$

cu exactitatea $\varepsilon>0$, utilizând în acest scop o formulă de integrare numerică.

Noțiunea de integrală definită de la funcția $f(x)$, definită pe intervalul $[a;b]$, se va înțelege în sensul lui Riemann. Din cursul *Analiza matematică* se ştie că aceasta reprezintă un număr ce poate fi calculat, când se cunoaşte primitiva $F(x)$ a integrandului, cu ajutorul formulei Leibniz-Newton

$$I=F(b)-F(a). \tag{3.2}$$

O condiție suficientă pentru ca să existe integrala definită (3.1) este ca funcția $f(x)$ să fie continuă pe $[a;b]$. Dar primitiva ei, deseori, nu se exprimă prin funcții elementare, ceea ce face imposibilă aplicarea formulei (3.2) în acest caz. În afară de aceasta, în multe probleme cu caracter aplicativ, funcția $f(x)$ nu se cunoaşte ea însăşi decât sub formă tabelară şi atunci determinarea primitivei în genere nu poate fi făcută. Din aceste motive formula (3.2) nu are aplicare largă în practică şi se recurge frecvent la calculul aproximativ al integralei cu ajutorul formulelor de cuadratură (prescurtat FC). Forma generală a acestor formule este [2-5; 8; 10; 13; 15; 16; 20-21; 23-24]

$$I_n=\sum_{i=0}^{n}c_i f(x_i), \tag{3.3}$$

unde $x_i\in[a;b]$ $(i=\overline{0,n})$ şi se numesc nodurile formulei de integrare numerică, iar c_i $(i=\overline{0,n})$ sunt coeficienții ei. Restul $R_n(f)$ al FC (3.3) este definit ca $R_n(f):=I-I_n$. Acesta oferă informații privind clasa funcțiilor pentru care formula de integrare numerică este eficientă, în sensul că pentru funcția dată şi $\varepsilon>0$, pentru numere n suficient de mari, are loc relația $|R_n(f)|=|I-I_n|<\varepsilon$. În aplicații, testarea acurateței aproximării impune verificarea unei inegalități de forma $|I_{n'}-I_n|<\varepsilon$ pentru $n'>n$.

Trebuie menționat, de asemenea, că şi pentru calculul integralei $\int_a^b\rho(x)f(x)dx$, în care $\rho:(a;b)\to\mathbb{R}$ este o funcție continuă pozitivă, numită pondere, pot fi construite FC de forma (3.3).

O metodă de obținere a unor FC constă în aproximarea funcției $f(x)$ prin polinomul de interpolare $P_n(x)$, construit pe sistemul de noduri $x_0,x_1,...,x_n$ de pe $[a;b]$ şi integrarea (analitic) celui din urmă în aceleaşi limite. Diferite variante de formule se pot obține în dependență de ponderea concretă $\rho(x)$, de

numărul de noduri n, de alegerea concretă a nodurilor x_i, cât şi de forma polinomului de interpolare ales pentru aproximare.

Mai întâi considerăm cazul $\rho(x)=1$. Împărțim segmentul $[a;b]$ în n părți egale, de lungime $h=(b-a)/n$, prin intermediul punctelor echidistante

$$x_i = a+ih,\ i=\overline{0,n}. \tag{3.4}$$

♦ **Formula dreptunghiurilor**

Dacă funcția $f(x)$ este aproximată pe $[a;b]$ cu un polinom de gradul zero $P_0(x)=f(\overline{x})$, unde $\overline{x}$ este mijlocul intervalului de integrare, rezultă formula dreptunghiului:

$$I=(b-a)\,f\left((a+b)/2\right). \tag{3.5}$$

În practică această formulă se aplică sub formă generalizată. Integrala definită pe $[a;b]$ se exprimă ca suma integralelor pe fiecare din subintervalele $[x_i;x_{i+1}],\ i=\overline{0,n-1}$ (valorile $x_i,\ i=\overline{0,n}$ sunt definite de relația (3.4)), apoi la fiecare termen al sumei se aplică formula dreptunghiului. Astfel obținem relația:

$$I=h\sum_{i=0}^{n-1} f(x_i+h/2), \tag{3.6}$$

care poartă denumirea de formula generalizată a dreptunghiurilor medii. În cazul în care $f(x)\in C^2[a;b]$, estimarea restului $R_n^d(f)$ al formulei (3.6) poate fi făcută cu ajutorul inegalității [1; 6]:

$$\left|R_n^d(f)\right|\le\left((b-a)^3M_2\right)\Big/\left(24n^2\right),\ M_2:=\max_{x\in[a;b]}\left|f''(x)\right|. \tag{3.7}$$

♦ **Formula trapezelor**

Această formulă se obține dacă funcția $f(x)$ se aproximează pe $[a;b]$ prin polinomul de interpolare de gradul întâi $P_1(x)$ conform nodurilor $x_0=a,\ x_1=b$ şi are forma:

$$I=(b-a)\left(f(a)+f(b)\right)/2. \tag{3.8}$$

Utilizând proprietatea de aditivitate a integralei față de intervalul de integrare şi formula (3.8) pentru fiecare din subintervalele $[x_i;x_{i+1}]$ $(i=\overline{0,n-1})$, se obține formula generalizată a trapezelor:

$$I=h\left(\left(f(x_0)+f(x_n)\right)/2+\sum_{i=1}^{n-1} f(x_i)\right). \tag{3.9}$$

În cazul în care $f(x)\in C^2[a;b]$, pentru restul $R_n^t(f)$ al FC (3.9) avem estimația [2-8; 10; 16; 20-21; 23-25]

$$\left|R_n^t(f)\right|\le\left((b-a)^3M_2\right)\Big/\left(12n^2\right),\ M_2=\max_{x\in[a;b]}\left|f''(x)\right|. \tag{3.10}$$

♦ **Formula lui Simpson (parabolelor)**

Considerând că funcția de integrat $f(x)$ se aproximează pe $[a;b]$ prin polinomul de interpolare de gradul doi $P_2(x)$, conform nodurilor $x_0=a,\ x_1=(a+b)/2,\ x_2=b$, obținem următoarea aproximație a integralei căutate

$$I=\left(f(a)+4f\left((a+b)/2\right)+f(b)\right)(b-a)/6, \tag{3.11}$$

ce poartă numele lui Simpson. Pentru a obține şi în acest caz o formulă, care să asigure o precizie mai

mare, se consideră că segmentul $[a;b]$ este divizat în $n=2m$ părți egale cu ajutorul punctelor (3.4). Pentru fiecare subinterval de forma $[x_i;x_{i+2}]$ aplicăm formula (3.11) şi sumăm după toate valorile lui $i=\overline{0,n-2}$. Drept rezultat obținem formula generalizată a lui Simpson:

$$I=\frac{h}{3}\left(f(x_0)+f(x_n)+4\sum_{i=1}^{n/2}f(x_{2i-1})+2\sum_{i=1}^{n/2-1}f(x_{2i})\right). \tag{3.12}$$

În cazul în care $f(x)\in C^{(IV)}[a;b]$, pentru restul $R_n^t(f)$ al formulei (3.12) avem estimația [2-6; 16; 24]

$$\left|R_n^S(f)\right|\le\left((b-a)^5M_4\right)\Big/\left(180n^4\right),\ M_4=\max_{x\in[a;b]}\left|f^{(IV)}(x)\right|. \tag{3.13}$$

Pseudocodul algoritmilor

Date de intrare:
a,b (de tip real) – limitele intervalului de integrare;
n (de tip întreg) – numărul de părți în care este divizat intervalul de integrare;
$f(x)$ - numele subprogramului-funcție (de tip real), cu ajutorul căruia se calculează valoarea integrandului $f(x)$.
Date de ieşire:
s (de tip real) – valoarea aproximativă a integralei;
întreg i
real h,s,x

formula dreptunghiurilor	*formula trapezelor*
$h:=(b-a)/n;s:=0;x:=a+h/2$ pentru $i:=\overline{0,n-1}$ execută $s:=s+f(x+ih)$ $s:=hs$ scrie s	$h:=(b-a)/n;s:=0$ pentru $i:=\overline{1,n-1}$ execută $s:=s+f(a+ih)$ $s:=h(s+0.5(f(a)+f(b)))$ scrie s

formula lui Simpson

```
h:=(b-a)/n; s:=0.5(f(a)-f(b)); i:=1
execută
 {
   x:=a+ih; s:=s+2f(x)+f(x+h); i:=i+2
 }
cât timp i<n
s:=2hs/3
scrie s
```

Remarcă. Adunarea repetată a lungimii pasului sub forma $x_i=x_{i-1}+h$, în cazul în care se generează un număr mare de noduri de integrare, poate condiționa acumularea de erori. Drept urmare, în descrierea algoritmilor s-a utilizat relația $x_i=a+ih,\ i=\overline{0,n}$.

♦ **Algoritmi cu control automat al pasului de integrare (implicit şi al exactității de calcul)**

Evaluarea constantelor M_2 şi M_4, prezente în relațiile (3.7), (3.10) şi (3.13), de cele mai dese ori întâmpină dificultăți serioase. De aceea, în practică controlul preciziei de integrare se efectuează în baza unui algoritm numeric, cunoscut ca *procedeul lui Runge* [25, p.149]. Acest proces implică la pasul k $(k=0,1,...)$ calculul aproximativ al integralei (3.1) conform uneia dintre formulele (3.6), (3.9) sau (3.12), pentru o valoare h a pasului, corespunzând unui număr $p=2^k+1$ de puncte de integrare, şi compararea valorii $I^{(k)}$ astfel obținute cu valoarea $I^{(k+1)}$ rezultată pentru pasul $h/2$, corespunzând unui număr de $2p-1$ puncte de integrare. Întrucât pentru eroarea de aproximare a integralei (3.1) are loc

următoarea relație [16, p.394]

$$\left|I - I^{(k+1)}\right| \simeq \left|I^{(k+1)} - I^{(k)}\right| \Big/ \left(2^m - 1\right),$$

(m este o constantă care coincide cu ordinul formulei de cuadratură aplicate) înjumătățirea pasului trebuie continuată, în principiu, până când eroarea $\left|I^{(k+1)} - I^{(k)}\right|$ pentru două aproximații consecutive ale integralei scade sub o valoare admisibilă $\varepsilon(2^m - 1)$. Apoi se consideră $I \simeq I^{(k+1)}$. Trebuie menționat că pentru formulele (3.6) şi (3.9) avem $m = 2$, iar pentru (3.12) - $m = 4$.

Calculul integralei cu o precizie prestabilită prin metoda înjumătățirii pasului poate fi optimizat observând că nu este necesară evaluarea funcției în toate punctele de integrare, ci doar în cele prin care au fost înjumătățite subintervalele rețelei utilizate în etapa anterioară. Restul valorilor funcției, deja calculate la paşii anteriori, sunt folosite implicit stabilind o relație de recurență între două aproximații consecutive ale integralei.

Pseudocodul algoritmilor

Cazul formulei dreptunghiurilor generalizate

Date de intrare: a, b (de tip real) – limitele intervalului de integrare; $f(x)$ - numele subprogramului-funcție (de tip real), cu ajutorul căruia se calculează valoarea integrandului $f(x)$; ε (de tip real) - exactitatea de calcul. Date de ieşire: $s2$ (de tip real) – valoarea aproximativă a integralei; k (de tip întreg) – număr ce indică de câte ori a fost înjumătățit pasul de integrare h; întreg n, i, k real $h, s1, s2, x, \varepsilon_1$

$k := 0;\ \varepsilon_1 := 3\varepsilon;\ n := 1;\ h := b - a;\ s2 := h\, f((a+b)/2)$
executată
{
$n := 2n;\ s1 := s2;\ h := h/2;\ k := k+1;\ s2 := 0;\ x := a - h/2$
pentru $i := \overline{1,n}$ execută $s2 := s2 + f(x + ih)$
$s2 := s2 \cdot h$
}
cât timp $\left\|s2 - s1\right\| \geq \varepsilon_1$
scrie $s2, k$

Cazul formulei trapezelor generalizate

Se poate verifica fără dificultate că succesiunea de aproximații obținute prin formula (3.9) poate fi descrisă cu ajutorul procesului iterativ [3, p.76]:

$$\begin{gathered} I_{tr}^{(0)} = (f(a) + f(b))h_0/2,\ h_0 = b - a,\ n_0 = 1, \\ I_{tr}^{(k)} = \frac{1}{2}\left(I_{tr}^{(k-1)} + h_{k-1} \sum_{i=1}^{n_{k-1}} f(a + (i - 1/2)h_{k-1}) \right),\ h_k = h_{k-1}/2,\ n_k = 2n_{k-1},\ k = 1, 2, \ldots, \end{gathered} \tag{3.14}$$

unde $h_k = (b-a)/n_k$ este lungimea subintervalelor după etapa k, iar $n_k = 2^k$ reprezintă numărul corespunzător de subintervale sau, echivalent, numărul punctelor noi de integrare şi de înjumătățire a subintervalelor pentru etapa următoare. Procesul recurent (3.14) trebuie continuat până când modulul diferenței dintre două aproximații succesive ale integralei devine mai mic decât $\varepsilon(2^m - 1)$.

Date de intrare:
a, b (de tip real) – limitele intervalului de integrare;
$f(x)$ - numele subprogramului-funcție (de tip real), cu ajutorul căruia se calculează valoarea integrandului $f(x)$;
ε (de tip real) - exactitatea de calcul.
Date de ieşire:
$t2$ (de tip real) – valoarea aproximativă a integralei;
k (de tip întreg) – număr ce indică de câte ori a fost înjumătățit pasul de integrare h;
întreg n, i, k
real $h, t1, t2, x, \varepsilon_1$

$k := 0;\ \varepsilon_1 := 3\varepsilon;\ n := 1;\ h := b - a;\ t2 := 0.5h(f(a) + f(b))$
execută
{
$t1 := t2;\ t2 := 0;\ x := a - 0.5h$
pentru $i := \overline{1,n}$ execută $t2 := t2 + f(x + ih)$
$t2 := 0.5(t1 + h \cdot t2);\ n := 2n;\ h := 0.5h;\ k := k + 1$
}
cât timp $|t2 - t1| \geq \varepsilon_1$
scrie $t2, k$

Cazul formulei lui Simpson generalizate

Având în vedere aproximațiile date de formula generalizată a trapezelor pentru valori ale pasului de integrare înjumătățite succesiv, se poate verifica că aproximațiile formulei generalizate a lui Simpson pot fi exprimate în raport cu cele dintâi:

$$I_S^{(k)} = (4I_{tr}^{(k)} - I_{tr}^{(k-1)})/3.$$

Atunci, algoritmul pentru calculul recursiv al aproximațiilor conform formulei (3.12) poate fi descris cu ajutorul relațiilor:

$$I_{tr}^{(0)} = (f(a) + f(b))h_0/2,\ h_0 = b - a,\ n_0 = 1,$$
$$I_{tr}^{(k)} = \frac{1}{2}\left(I_{tr}^{(k-1)} + h_{k-1}\sum_{i=1}^{n_{k-1}} f(a + (i - 1/2)h_{k-1}) \right),\ k = 1, 2, ...,$$
$$I_S^{(k)} = (4I_{tr}^{(k)} - I_{tr}^{(k-1)})/3;\ h_k = h_{k-1}/2,\ n_k = 2n_{k-1}\ \ (h_k = (b-a)/n_k,\ n_k = 2^k).$$

Date de intrare:
a, b (de tip real) – limitele intervalului de integrare;
$f(x)$ - numele subprogramului-funcție (de tip real), cu ajutorul căruia se calculează valoarea integrandului $f(x)$;
ε (de tip real) - exactitatea de calcul.
Date de ieşire:
$s2$ (de tip real) – valoarea aproximativă a integralei;
k (de tip întreg) – număr ce indică de câte ori a fost înjumătățit pasul de integrare h;
întreg n, i, k
real $h, t1, t2, s1, s2, \varepsilon_1$

$k := 0;\ \varepsilon_1 := 15\varepsilon;\ n := 1;\ h := b - a;\ t2 := 0.5h(f(a) + f(b));\ s2 := t2$
execută
{
$t1 := t2;\ t2 := 0;\ s1 := a - 0.5h$
pentru $i := \overline{1,n}$ execută $t2 := t2 + f(s1 + ih)$
$s1 := s2;\ t2 := 0.5(t1 + h \cdot t2);\ s2 := (4 \cdot t2 - t1)/3;\ n := 2n;\ h := 0.5h;\ k := k + 1$

```
    }
  cât timp |s2 − s1| ≥ ε1
scrie  s2, k
```

Remarcă. Pentru ca numărul n al divizărilor intervalului $[a;b]$, ca putere a lui doi, să nu depăşească valoarea maximă de tip *long* reprezentabilă, se recomandă ca numărul de înjumătăţiri ale subintervalelor să fie limitat. Dacă, datorită convergenței slabe a procesului, se epuizează numărul maximal admisibil de iterații posibile fără a se fi atins exactitatea dorită în calculul integralei (3.1), se iese din ciclul înjumătăţirilor şi se emite un mesaj de eroare.

Analiza algoritmilor

Se observă că restul formulei (3.12) depinde de h^4 (eroarea formulei este de ordinul patru $R_n^S(f) = O(h^4)$), în timp ce restul formulelor (3.6) şi (3.9) depinde doar de h^2. Astfel, formula lui Simpson este cu un grad de precizie mai ridicat decât formulele dreptunghiului sau trapezului. În schimb, acestea din urmă sunt mai flexibile, deoarece utilizează o rețea cu un număr oarecare de noduri, spre deosebire de formula lui Simpson, care necesită un număr impar de noduri. Formulele (3.6) şi (3.9) sunt exacte pentru orice funcție liniară, pe când formula (3.12) este exactă pentru orice polinom de grad mai mic sau egal cu trei. FC (3.6), (3.9) şi (3.12) sunt cazuri particulare ale formulelor Newton-Côtes [2, p.88].

♦ **Formule de cuadratură de tip Gauss**

Spre deosebire de FC Newon-Côtes, care utilizează diviziuni echidistante ale intervalului de integrare, în formulele de tip Gauss nodurile se aleg într-un mod optim pentru ca eroarea de aproximare a integralei să fie minimă.

În cele ce urmează vom considera formule de integrare numerică de forma:

$$\int_a^b \rho(x) f(x)dx \simeq \sum_{i=1}^{n} c_i f(x_i), \tag{3.15}$$

unde ponderea $\rho(x)$ este o funcție continuă şi pozitivă pe $(a;b)$. Dacă nodurile x_i $(i=\overline{1,n})$ se aleg ca fiind zerourile unui polinom $q(x)$ de gradul n ce satisface relația de ortogonalitate

$$\int_a^b \rho(x) p(x) q(x)dx = 0, \tag{3.16}$$

pentru orice polinom $p(x)$ de grad până la $n-1$, atunci formula (3.15) are gradul (maxim) de exactitate $2n-1$ (formula va fi exactă pentru orice polinom de grad mai mic sau egal cu $2n-1$) [3, p.70]. FC de tip Gauss sunt formule de forma (3.15) pentru care se atinge gradul maxim de exactitate. Zerourile unui polinom ce satisface relația (3.16) sunt simple şi aparțin intervalului deschis $(a;b)$.

Odată stabilită familia de polinoame ortogonale în raport cu ponderea $\rho(x)$, se află rădăcinile acestora, iar coeficienții c_i $(i=\overline{0,n})$ rezultă din formula $c_i = \int_a^b \rho(x) l_i(x)dx$, în care $l_i(x)$ sunt polinoamele fundamentale de interpolare Lagrange.

Pentru orice funcție continuă pe $[a;b]$, aproximațiile integralei conform formulei de tip Gauss vor

converge când $n \to \infty$ către valoarea exactă a integralei [4, p.174].

Cazul $\rho(x)=1$. Formula de cuadratură a lui Gauss-Legendre

Vom analiza mai întâi cum se aproximează integrala $\int_{-1}^{1} f(t)dt$ şi vom extinde apoi metoda pentru calculul integralei pe orice interval $[a;b]$.

Se pune problema de a determina nodurile t_i $(i=\overline{1,n})$ din intervalul $[-1;1]$ şi coeficienții c_i $(i=\overline{1,n})$ astfel încât FC $\int_{-1}^{1} f(t)dt \simeq \sum_{i=1}^{n} c_i f(t_i)$ să fie exactă pentru orice polinom de grad mai mic sau egal cu $2n-1$.

Pentru un n fixat în calitate de noduri ale FC se iau zerourile polinomului de gradul n din sistemul ortogonal a lui Legendre:

$$L_n(t)=\frac{1}{2^n n!}\frac{d^n}{dt^n}\left((t^2-1)^n\right),\ n=0,1,...\ \ (t\in[-1;1]).$$

Polinomul $L_n(t)$ are n rădăcini reale, distincte, simetrice față de origine, situate în intervalul $(-1;1)$.

Coeficienții c_i se obțin prin integrarea pe intervalul $[-1;1]$ a polinoamelor fundamentale de interpolare Lagrange, construite cu ajutorul rădăcinilor polinomului Legendre $L_n(t)$:

$$c_i=\int_{-1}^{1}\left(\prod_{j=1,\,j\neq i}^{n}\left(\left(t-t_j\right)/\left(t_i-t_j\right)\right)\right)dt,\ i=\overline{1,n}.$$

În general, pentru calculul aproximativ al integralei $\int_a^b f(x)dx$ se face mai întâi schimbarea de variabilă $x=t(b-a)/2+(b+a)/2$, pentru a avea limitele de integrare -1 şi 1. Astfel, se obține formula $\int_a^b f(x)dx=\frac{b-a}{2}\int_{-1}^{1}F(t)dt,\ F(t)=f\left(t(b-a)/2+(b+a)/2\right)$. Aplicând ultimei integrale FC a lui Gauss–Legendre, avem:

$$\int_a^b f(x)dx \simeq \frac{b-a}{2}\sum_{i=1}^{n} c_i f(x_i), \tag{3.17}$$

unde $x_i=t_i(b-a)/2+(b+a)/2,\ i=\overline{1,n}$, cu t_i zerourile din intervalul $(-1;1)$ ale polinomului Legendre $L_n(t)$.

Există tabele speciale unde pot fi găsiți coeficienții şi nodurile FC a lui Gauss-Legendre pentru diferite valori ale lui n. În tabelul de mai jos sunt indicate nodurile de integrare t_i şi coeficienții c_i corespunzători, pentru un număr de intervale $n=2,3,4,5$.

n	$t_i,\ i=\overline{1,n}$	$c_i,\ i=\overline{1,n}$
2	$t_2=-t_1=1/\sqrt{3}$	$c_1=c_2=1$
3	$t_3=-t_1=\sqrt{3/5},\ t_2=0$	$c_1=c_3=5/9,\ c_2=8/9$
4	$t_4=-t_1=0.8611363116,\ t_3=-t_2=0.3399810436$	$c_1=c_4=0.3478548451,\ c_2=c_3=0.6521451549$

5	$t_5 = -t_1 = 0.9061798459,\ t_4 = -t_2 = 0.5384693101,$ $t_3 = 0$	$c_1 = c_5 = 0.2369268850,\ c_2 = c_4 = 0.4786286705,$ $c_3 = 0.5688888889$

Cazul $a = -1, b = 1, \rho(t) = 1/\sqrt{1-t^2}$ **. Formula de cuadratură a lui Gauss-Chebyshev**

Se poate observa că funcția pondere $\rho(t)$ prezintă singularități în ± 1. Utilitatea practică a unei astfel de ponderi este integrarea numerică a funcțiilor ce prezintă acelaşi gen de singularitate integrabilă în capetele domeniului de integrare, utilizând procedeul izolării comportării singulare din capete în funcția pondere.

FC a lui Gauss-Chebyshev se aplică pentru integrale de forma $\int_{-1}^{1}\left(1-t^2\right)^{-1/2} F(t)dt$ şi are aspectul:

$$\int_{-1}^{1}\frac{1}{\sqrt{1-t^2}}F(t)dt \simeq \sum_{i=1}^{n} c_i F(t_i)\,. \tag{3.18}$$

Aici valorile t_i $(i=\overline{1,n})$ ale nodurilor de integrare sunt zerourile polinomului $T_n(t) = \cos(n\arccos(t))$ de gradul n din sistemul de polinoame ortogonale ale lui Chebyshev de speța întâi:

$$t_i = \cos\left(\left((2i-1)\pi\right)/\left(2n\right)\right),\ i=\overline{1,n}\,,$$

iar coeficienții c_i satisfac relația $c_i = \pi/n,\ i=\overline{1,n}$. Deoarece toți coeficienții c_i din formula (3.18) au aceeaşi valoare, această formulă necesită un număr de operații mai mic pentru atingerea unui anumit ordin de precizie, în comparație cu formula (3.17).

Orice integrală de forma $\int_a^b\left(1-x^2\right)^{-1/2} f(x)dx$ sau $\int_a^b f(x)dx$ $(-\infty < a < b < \infty)$, poate fi adusă la forma (3.18) printr-o transformare convenabilă de variabilă.

Pentru a calcula integrala $\int_a^b \rho(x) f(x)dx$ cu exactitatea dată $\varepsilon > 0$, conform formulei (3.17) sau (3.18), se poate aplica principiul lui Runge. Algoritmii de calcul descrişi de aceste două formule sunt simpli şi se implementează analog celor definiți de formulele (3.5), (3.8), (3.11).

3.III. Modul de desfăşurare a lucrării

♦ Cerințe față de îndeplinirea lucrării

Să se elaboreze un program destinat calculului integralei (3.1) cu exactitatea dată $\varepsilon > 0$. În acest scop, se vor implementa algoritmii metodelor numerice analizate la punctul 3.II într-un limbaj de programare de nivel înalt. Programul de calcul să se testeze pe exemplele propuse la sfârşitul lucrării de laborator.

Să se extragă la tipar valoarea aproximativă a integralei, numărul de înjumătățiri ale pasului inițial de integrare necesar pentru a obține exactitatea cerută $\varepsilon > 0$, precum şi timpii necesari pentru efectuarea unei iterații şi pentru atingerea unei precizii dorite. În plus, pentru fiecare algoritm se va afişa şi reprezenta grafic variația erorii aparente $\left|I^{(k+1)} - I^{(k)}\right|$ (distanța dintre două iterații succesive în cadrul procedeului lui Runge) pe parcursul iterațiilor. Se vor compara rezultatele obținute prin utilizarea

diferiților algoritmi (eficiența lor relativă în rezolvarea problemei date) şi se va determina algoritmul optim.

♦ **Chestiuni de studiat**

1. Comparați eficiența diferiților algoritmi pentru calculul aproximativ al aceleiaşi integrale.
2. Să se facă o analiză experimentală a erorii la integrarea numerică în funcție de valoarea pasului de integrare, precum şi de modul de alegere a nodurilor şi a coeficienților FC.
3. Să se facă o analiză experimentală a dependenței timpului de calcul a aproximației integralei de exactitatea de calcul impusă.

♦ **Probleme propuse spre rezolvare**

a) Pentru integralele definite ce urmează

1. $\int_0^1 \frac{1}{1+x^2}dx;$	7. $\int_0^2 \sin x^2 dx;$	13. $\int_0^1 \sqrt{x}\cos x dx;$
2. $\int_{-1}^1 \frac{e^x}{1+x^2}dx;$	8. $\int_0^\pi \sqrt{x+1}\sin x dx;$	14. $\int_0^\pi \frac{1}{1+x+\sqrt{\sin x}}dx;$
3. $\int_{-1}^1 e^{-x^2/2}dx;$	9. $\int_0^1 \sqrt{tgx+1}dx;$	15. $\int_0^{\pi/2} \frac{1}{x+\sqrt{\cos x}}dx;$
4. $\int_{-1}^1 \frac{(x^2+1)}{2-\sin x}dx;$	10. $\int_0^2 \frac{\ln\sqrt{x+1}}{\sqrt{x+1}}dx;$	16. $\int_{\pi/4}^{\pi/2} \frac{\sin x}{x}dx;$
5. $\int_0^2 \frac{x\cos x}{x+1}dx;$	11. $\int_0^{\pi/3} \frac{1}{\sqrt{1-0.25\sin^2 x}}dx;$	17. $\int_1^3 \frac{\lg x}{x}dx;$
6. $\int_{-1}^1 \frac{e^{-x}}{x+2}dx;$	12. $\int_0^{0.5} \sqrt{\frac{1-0.25x^2}{1-x^2}}dx;$	18. $\int_0^2 \frac{e^{-x^2}\sin 0.5x}{0.5+x^2}dx,$

aplicați una dintre formulele (3.6), (3.9), (3.12) pentru a calcula valoarea aproximativă a lor cu o eroare ce nu depăşeşte $\varepsilon = 10^{-6}$

b) Pentru următoarele integrale

19. $\int_{-1}^1 \frac{x\sin x}{\sqrt{1-x^2}}dx;$	21. $\int_0^1 \frac{2\sin x}{\sqrt{x(1-x)}}dx;$	23. $\int_{-1}^1 \frac{\sin x}{\sqrt{1+x^2}}dx;$	25. $\int_{-2}^2 \frac{\ln(1+x^2)}{\sqrt{(1-x)}}dx$
20. $\int_{-1}^1 \frac{\sqrt{\cos x^2}}{\sqrt{1-x^2}}dx;$	22. $\int_0^1 \frac{e^{2x-5}}{\sqrt{x(1-x)}}dx;$	24. $\int_{-1}^1 \frac{e^{x^2}}{1-x}dx;$	26. $\int_0^2 \frac{x^2 e^x}{1+x}dx,$

aplicați una dintre formulele de cuadratură a lui Gauss (3.17) sau (3.18), pentru a calcula valoarea lor aproximativă, folosind două valori succesive ale lui n. Eroarea se va determina prin compararea valorilor obținute.

c) Pentru integralele 23-26 din punctul b), să se calculeze valorile lor aproximative, aplicând formula Gauss (3.17) şi principiul lui Runge pentru a atinge precizia dată $\varepsilon = 10^{-5}$.

LUCRAREA DE LABORATOR №4
REZOLVAREA NUMERICĂ A ECUAȚIILOR ALGEBRICE ŞI TRANSCENDENTE

4.I. Scopul lucrării

Studiul unor metode numerice aproximative de rezolvare a ecuațiilor algebrice şi transcendente şi implementarea acestora într-un limbaj de programare de nivel înalt.

4.II. Prezentarea teoretică a lucrării

♦ Formularea problemei

Fie $f:[a;b]\to\mathbb{R}$ o funcție continuă şi derivabilă. Să se afle toate rădăcinile reale de pe $[a;b]$ ale ecuației

$$f(x)=0 \tag{4.1}$$

cu exactitatea dată $\varepsilon>0$, utilizând în acest scop o metodă aproximativă numerică (iterativă).

Problema formulată se rezolvă în două etape [2-5; 10; 13; 15-16; 20-21; 23-24; 26; 28]:

1) Se separă (localizează) rădăcinile, adică se stabilesc pe $[a;b]$ intervale care conțin fiecare una şi numai una dintre rădăcinile ecuației (4.1).

2) Pentru fiecare rădăcină separată se precizează aproximațiile ei până se ajunge la gradul de precizie dorit.

Vom analiza mai întâi cele mai răspândite metode de separare ale rădăcinilor ecuației de forma (4.1).

♦ Separarea rădăcinilor prin metoda tabelării funcției (metoda probelor)

Se determină semnele valorilor funcției $f(x)$ în nodurile rețelei echidistante $\alpha_i=\alpha_0+ih,\ i=\overline{0,n}$, $\alpha_0=a,\ h=(b-a)/n$. Dacă pentru nodurile vecine α_i şi α_{i+1} avem $f(\alpha_i)f(\alpha_{i+1})<0$, atunci pe segmentul $[\alpha_i;\alpha_{i+1}]$ se găseşte cel puțin o rădăcină a ecuației (4.1), iar în cazul când $f(\alpha_i)f(\alpha_{i+1})>0$, pe $[\alpha_i;\alpha_{i+1}]$ sau nu există rădăcini, sau există un număr par. Dacă nodurile rețelei sunt suficient de apropiate, atunci, cel mai probabil, când $f(\alpha_i)f(\alpha_{i+1})<0$, pe $[\alpha_i;\alpha_{i+1}]$ avem o singură rădăcină; iar când $f(\alpha_i)f(\alpha_{i+1})>0$, pe $[\alpha_i;\alpha_{i+1}]$ nu avem rădăcini (desigur, în acest caz excludem din studiu rădăcinile de multiplicitate pară). Deoarece siguranța metodei depinde de caracterul funcției $f(x)$, se recomandă, totuşi, să ne convingem suplimentar de aceasta, fie studiind semnul derivatei $f'(x)$ pe $[\alpha_i;\alpha_{i+1}]$ (în cazul când $f(\alpha_i)f(\alpha_{i+1})<0$, iar $f'(x)$ păstrează semnul, avem o singură rădăcină), fie studiind graficul funcției $f(x)$ pe acest interval.

Pseudocodul algoritmului

Date de intrare:
- f - numele subprogramului-funcție (de tip real), cu ajutorul căruia se calculează valoarea funcției $f(x)$;
- a,b (de tip real) – extremitățile segmentului pe care se testează existența rădăcinilor ecuației (a < b);
- n (de tip întreg) – numărul de puncte ale rețelei.

Date de ieşire:
- $c[]$ (tablou unidimensional cu componente de tip real) - tablou ce conține valorile extremităților de stânga ale intervalelor în care s-a depistat existența rădăcinii;

real $h, x_1, x_2\ f_1, f_2$
întreg i
tablou real $c[]$

$i := 0;\ h := (b-a)/n;\ x_1 := a;\ x_2 := x_1 + h;\ f_1 := f(x_1)$
cât timp $x_2 < b$ execută
{
$f_2 := f(x_2)$
dacă $f_1 f_2 < 0$ atunci { $c_i := x_1;\ i := i+1$ }
$x_1 := x_2;\ x_2 := x_1 + h;\ f_1 := f_2$
}

♦ **Separarea grafică a rădăcinilor**

Se construieşte graficul funcției $f(x)$ pe $[a;b]$ şi vizual se caută punctele de intersecție ale graficului cu axa absciselor. Simplu se stabilesc intervalele de separare a rădăcinilor. Construirea graficului deseori permite evidențierea rădăcinilor de multiplicitate pară. Deseori se aplică şi următorul procedeu: ecuația (4.1) se înlocuieşte cu o ecuație echivalentă ei, de forma $\varphi(x) = \psi(x)$, în care funcțiile $\varphi(x)$ şi $\psi(x)$ au reprezentări grafice mai simple. Abscisele punctelor de intersecție ale graficelor funcțiilor $\varphi(x)$ şi $\psi(x)$ sunt rădăcini ale ecuației (4.1).

În cele ce urmează vom prezenta câteva metode iterative pentru aproximarea rădăcinilor ecuației (4.1), metode care presupun că aceste rădăcini au fost în prealabil separate.

♦ **Metoda dihotomiei (bisecției, înjumătățirii intervalului)**

Fie $x^* \in [a_0;b_0] \subseteq [a;b]$ una dintre rădăcinile separate. Se generează un şir de intervale $[a_1;b_1], [a_2;b_2], \ldots$ cu proprietățile următoare:

a) $[a_0;b_0] \supset [a_1;b_1] \supset [a_2;b_2] \supset \ldots \supset [a_i;b_i] \supset \ldots$; b) $x^* \in [a_k;b_k],\ k = 1,2,\ldots$; c) $f(a_k)\, f(b_k) < 0$.

Intervalul $[a_k;b_k],\ k = 1,2,\ldots$ se obține în modul următor:

Bazându-ne pe observația că $f(x)$ are semne opuse pe extremitățile oricărui interval în interiorul căruia se conține rădăcina (unică), se înjumătățeşte intervalul $[a_{k-1};b_{k-1}]$ şi se selectează din cele două subintervale obținute, cel care conține rădăcina.

Şirul de aproximații $c_{k+1} = (a_k + b_k)/2$, $[a_{k+1};b_{k+1}] = \begin{cases} [a_k;c_{k+1}], \text{ dacă } f(a_k)\, f(c_{k+1}) < 0 \\ [c_{k+1};b_k], \text{ dacă } f(a_k)\, f(c_{k+1}) > 0 \end{cases}$, $k = 0,1,\ldots,$

converge în mod garantat către rădăcina ecuației (4.1) (evident, procesul descris se întrerupe dacă $f(c_{k+1}) = 0$ şi atunci se pune $x^* = c_{k+1}$). Dacă $f(c_k) \neq 0, k = 1,2,\ldots$, atunci continuăm calculul până când se va satisface inegalitatea $b_k - a_k < 2\varepsilon$ [10; 16; 21; 23-24]. Mijlocul $c_{k+1} = (a_k + b_k)/2$ al segmentului $[a_k;b_k]$ va aproxima soluția x^* cu exactitatea dorită ε.

Deoarece şirul $\{a_k\}_{k \geq 0}$ formează un şir de aproximații prin lipsă ale rădăcinii x^*, valorile $f(a_k),\ k = 0,1,\ldots$ au acelaşi semn. Ținând cont de aceasta, la iterația cu numărul k a algoritmului nu se va calcula $f(a_k)$, ci se va utiliza valoarea $f(a_0)$ (aceasta nu influențează asupra semnului produsului $f(a_k)\, f(c_{k+1})$).

Pseudocodul algoritmului

Date de intrare:
f - numele subprogramului-funcție (de tip real), cu ajutorul căruia se calculează valoarea funcției $f(x)$;

a, b (de tip real) – extremităţile segmentului pe care este separată rădăcina; ε (de tip real) – precizia dorită; ε_1 (de tip real) – orice mărime mai mică decât ε_1 se consideră egală cu zero. Date de ieşire: c (de tip real) – aproximaţia rădăcinii ecuaţiei; fc (de tip real) – valoarea funcţiei f în punctul c; k (de tip întreg) – numărul de iteraţii efectuate; întreg s, k real c, fc

$k := 0$; $s := sign(f(a))$; $c := 0.5(a+b)$; $fc := f(c)$
cât timp $(\lvert fc\rvert \geq \varepsilon_1$ şi $\lvert b-a\rvert \geq 2\varepsilon)$ execută
{
dacă $sign(fc) = s$ atunci $a := c$
altfel $b := c$
$k := k+1$; $c := 0.5(a+b)$; $fc := f(c)$
}
scrie c, fc, k

Analiza algoritmului

Spre deosebire de alte metode iterative, metoda dihotomiei întotdeauna converge atunci când funcţia $f(x)$ este continuă şi pentru aplicarea ei nu este necesară o alegere specială a aproximaţiei iniţiale din $[a_0; b_0]$. Metoda converge liniar cu viteza unei progresii geometrice cu raţia $1/2$ [10; 16; 21; 23-24]. Numărul de iteraţii n necesare pentru atingerea exactităţii ε este $n = \left\lceil \log_2 \left((b_0 - a_0)/\varepsilon \right) \right\rceil$. De regulă, pentru metoda dihotomiei n este mai mare decât pentru alte metode iterative. Vom menţiona că metoda nu poate fi aplicată pentru a calcula rădăcinile de multiplicitate pară.

♦ **Metoda lui Newton (tangentelor)**

Conform metodei lui Newton aproximaţia x_{k+1} a rădăcinii este punctul de intersecţie al tangentei la curba $y = f(x)$, dusă în punctul $(x_k; f(x_k))$, cu axa absciselor. Ipotezele necesare pentru aplicarea metodei sunt următoarele [2-5]: $f \in C^2[a_0; b_0]$ (există derivatele $f'(x), f''(x)$ continui pe $[a_0; b_0]$), $f(a_0) f(b_0) < 0$, iar $f'(x)$ şi $f''(x)$ păstrează semnul constant pe $[a_0; b_0]$. Aceste condiţii asigură existenţa unei singure rădăcini x^* pe $[a_0; b_0]$. Putem concluziona că pentru a aplica cu succes garantat metoda lui Newton este necesar nu doar să se afle un careva interval de separare a rădăcinii, ci să se stabilească un astfel de interval (într-atât de mic) încât funcţia $f(x)$ să fie monotonă (adică $f'(x)$ păstrează semnul) şi convexă sau concavă (adică $f''(x)$ păstrează semnul) pe acesta.

Formula de calcul a metodei lui Newton este definită prin relaţia recurentă:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}, \; k = 0, 1, 2, \ldots, \tag{4.2}$$

În condiţiile impuse mai sus asupra funcţiei $f(x)$, dacă aproximaţia iniţială $x_0 \in [a_0; b_0]$ se alege astfel încât să se satisfacă condiţia:

$$f(x_0) f''(x_0) > 0, \tag{4.3}$$

atunci şirul numeric $\{x_k\}_{k \geq 0}$ generat conform formulei (4.2) va converge către rădăcina x^* [2-5; 8].

Drept criteriu de stopare a calculelor în acest proces iterativ serveşte inegalitatea [2-5; 8; 17]:

$$\left|x_{k+1}-x^*\right| \le \frac{M_2}{2m_1}\left(x_{k+1}-x_k\right)^2 < \varepsilon, \quad k=0,1,...; \quad M_2 = \max_{x\in[a_0;b_0]}\left|f''(x)\right|, \quad m_1 = \min_{x\in[a_0;b_0]}\left|f'(x)\right|. \tag{4.4}$$

În cazul în care calculul constantelor m_1 şi M_2 întâmpină dificultăți, pentru a calcula rădăcina ecuației (4.1) cu exactitatea ε poate fi utilizat următorul criteriu: procesul iterativ stopează atunci când ambele mărimi $\left|x_{k+1}-x_k\right|$ şi $\left|f(x_{k+1})\right|$ devin mai mici ca ε.

Pseudocodul algoritmului

Date de intrare:
f (f', f'') – numele subprogramului-funcție (de tip real), cu ajutorul căruia se calculează valoarea funcției $f(x)$ (derivatei $f'(x)$, derivatei a doua $f''(x)$);
a, b (de tip real) – extremitățile segmentului pe care este separată rădăcina;
ε (de tip real) – precizia dorită;
m_1, M_2 (de tip real) – constante ce figurează în formula (4.4) pentru stoparea calculelor.
Date de ieşire:
x (de tip real) – aproximația rădăcinii ecuației;
$f(x)$ (de tip real) – valoarea funcției f în punctul x;
k (de tip întreg) – numărul de iterații efectuate;
întreg k
real x, ε_1, d

$\varepsilon_1 := \sqrt{2m_1\varepsilon/M_2}$; $k := 0$
dacă $f(a)f''(a) > 0$ atunci $x := a$
altfel $x := b$
execută
{
$d := f(x)/f'(x)$; $x := x - d$; $k := k + 1$
}
cât timp $|d| \ge \varepsilon_1$
scrie $x, f(x), k$

Algoritmul lui Newton a fost descris pentru cazul în care rădăcina ecuației este separată pe intervalul $[a_0;b_0]$, iar funcțiile $f'(x)$ şi $f''(x)$ au semn constant pe acest interval, aproximația inițială x_0 alegându-se conform formulei (4.3). În cazul general, când este dificil să se separe rădăcina astfel încât să se satisfacă toate condițiile ce asigură convergența metodei, se procedează în modul următor: aproximația inițială se alege aleator de pe $[a_0;b_0]$, iar dacă după un număr fixat de iterații nu ajungem la precizia dorită, se reia procedeul cu o altă valoare a aproximației inițiale. În cazul menționat vom utiliza următorul algoritm de calcul:

Date de intrare:
f (f') – numele subprogramului-funcție (de tip real), cu ajutorul căruia se calculează valoarea funcției $f(x)$ (derivatei $f'(x)$);
x_0 (de tip real) – aproximația inițială;
n_{max} (de tip întreg) – numărul de iterații maximal admisibil;
ε (de tip real) – precizia dorită.
Date de ieşire:
x (de tip real) – aproximația rădăcinii ecuației;
$f(x)$ (de tip real) – valoarea funcției f în punctul x;
k (de tip întreg) – numărul de iterații efectuate;
ind (de tip întreg) – indicatorul cauzei stopării procesului de calcul:

dacă $ind = 0$, atunci rădăcina ecuației este găsită cu exactitatea dorită; dacă $ind = 1$, atunci iterațiile nu converg la rădăcina ecuației; dacă $ind = 2$, atunci numărul de iterații efectuate a depăşit numărul maximal admisibil $n_{\max}$, ceea ce înseamnă sau că s-a luat o aproximație inițială nepotrivită, sau că valoarea $\lvert f'(x)\rvert$ este mică în vecinătatea rădăcinii; întreg k, ind real x, d_0, d

$k := 0;\ d_0 := [\log_2 \log_2 1/\varepsilon] + 1;\ ind := 0;$
execută
{
$d := f(x_0)/f'(x_0);\ x := x_0 - d$
dacă $\lvert d\rvert \le d_0$ atunci
{
dacă $k > n_{\max}$ atunci { $ind := 2$; STOP }
$k := k + 1;\ x_0 := x;\ d_0 := \lvert d\rvert$
}
altfel { $ind := 1$; STOP }
}
cât timp $\lvert d\rvert \ge \varepsilon$
scrie $x, f(x), k, ind$

Analiza algoritmului

În cazul în care se satisfac condițiile suficiente de convergență, şirul $\{x_k\}_{k\ge 0}$, generat conform metodei lui Newton, converge monoton şi unilateral către rădăcina x^* [2-5; 16; 23-24]. Numărul minim de iterații n necesare pentru calculul rădăcinii cu exactitatea relativă ε este $n = [\log_2 \log_2 (1/\varepsilon)] + 1$ [17]. La fiecare iterație a metodei lui Newton volumul de calcule este mai mare decât la alte metode iterative, deoarece este necesar să se calculeze nu numai valoarea funcției $f(x)$, ci şi a derivatei ei. În schimb viteza de convergență este esențial mai mare decât la alte metode. Dacă rădăcina x^* este simplă, metoda lui Newton are viteză pătratică de convergență. Aceasta înseamnă că eroarea aproximației în vecinătatea rădăcinii simple este aproximativ egală pătratului erorii aproximației precedente, adică după fiecare iterație numărul cifrelor exacte se dublează.

Nu este indicat să aplicăm metoda lui Newton atunci când prima derivată are valori mici în vecinătatea rădăcinii, adică graficul funcției în vecinătatea rădăcinii este „aproape” paralel axei absciselor. În acest caz, procesul de calcul ar putea să devină laborios, iar erorile de rotunjire a numerelor pot genera efectul invers – şirul de aproximații va începe să diveargă.

♦ **Metoda lui Newton modificată (Newton-Kantorovich)**

Atunci când calculul valorilor $f'(x_k)$ din formula (4.2) implică dificultăți, se poate utiliza o variantă simplificată, cunoscută sub numele de Newton–Kantorovich [23, p.219]. Acest algoritm utilizează o singură evaluare a derivatei în punctul inițial x_0. Procesul iterativ se defineşte prin următoarea formulă de recurență:

$$x_{k+1} = x_k - f(x_k)/f'(x_0),\ k = 0,1,\ldots,$$

unde $x_0 \in [a_0; b_0]$ verifică condiția (4.3). Şirul $\{x_k\}_{k\ge 0}$ converge către rădăcina x^* a ecuației (4.1) în

aceleaşi condiţii ca şi pentru metoda lui Newton [23-24]. Pseudocodul algoritmului de calcul al metodei lui Newton se modifică uşor pentru rezolvarea ecuației (4.1), conform metodei lui Newton modificate.

Analiza algoritmului

Deşi converge mai lent decât metoda lui Newton, metoda modificată are avantajul că nu necesită calculul derivatei $f'(x_k)$ la fiecare iterație.

♦ **Metoda secantelor (Newton discretă)**

Pentru a evita calculul valorilor derivatei, în metoda lui Newton se poate face apel la aproximarea numerică a acestora:

$$f'(x_k) \approx \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}.$$

Metoda astfel obținută, bazată pe formula recurentă:

$$x_{k+1} = x_k - \frac{f(x_k) \cdot (x_k - x_{k-1})}{f(x_k) - f(x_{k-1})}, \quad k = 1, 2, \ldots, \tag{4.5}$$

este cunoscută sub numele de metoda secantelor. Aproximația x_{k+1} a rădăcinii este punctul de intersecție al secantei duse prin $(x_{k-1}; f(x_{k-1}))$ şi $(x_k; f(x_k))$ cu axa absciselor.

Metoda foloseşte o funcție de iterație de două variabile $x_{k+1} = g(x_{k-1}, x_k)$, deci, necesită o inițializare dublă x_0, x_1. Dacă asupra funcției $f(x)$ se impun aceleaşi restricții ca şi în cazul metodei lui Newton, aproximațiile inițiale x_0 şi x_1 se pot alege în modul următor:

$$x_0 = \begin{cases} a_0 \ \ dacă \ f(a_0) f''(a_0) > 0 \\ b_0 \ \ dacă \ f(a_0) f''(a_0) < 0 \end{cases}, \quad x_1 = \begin{cases} a_0 + \varepsilon \ \ dacă \ f(a_0) f''(a_0) > 0 \\ b_0 - \varepsilon \ \ dacă \ f(a_0) f''(a_0) < 0 \end{cases}.$$

Aici ε este exactitatea cu care se calculează rădăcina.

Întrucât şirul numeric generat conform metodei secantelor converge unilateral către rădăcină, calculele vor stopa atunci, când ambele mărimi $|x_{k+1} - x_k|$ şi $|f(x_{k+1})|$ devin mai mici ca ε [17; 21; 24].

Pseudocodul algoritmului

Date de intrare:
- f (f'') – numele subprogramului-funcție (de tip real), cu ajutorul căruia se calculează valoarea funcției $f(x)$ (derivatei a doua $f''(x)$);
- a, b (de tip real) – extremitățile segmentului pe care este separată rădăcina;
- ε (de tip real) – precizia dorită.

Date de ieşire:
- x (de tip real) – aproximația rădăcinii ecuației;
- fx (de tip real) – valoarea funcției f în punctul x;
- k (de tip întreg) – numărul de iterații efectuate;

întreg k
real x_0, x, d, fx_0, fx

dacă $f(a) f''(a) > 0$ atunci $\{x_0 := a;\ x := a + \varepsilon\}$
altfel $\{x_0 := b;\ x := b - \varepsilon\}$
$k := 0;\ d := x - x_0;\ fx_0 := f(x_0);\ fx := f(x)$
executată
{
$d := d \cdot fx / (fx_0 - fx);\ x := x + d;\ fx_0 := fx;\ fx := f(x);\ k := k + 1$

```
    }
  cât timp (|d| ≥ ε sau |fx| ≥ ε)
scrie x, fx, k
```

Algoritmul de calcul a fost descris astfel, încât la fiecare iterație să se calculeze o singură valoare nouă pentru funcția $f(x)$.

Analiza algoritmului

Metoda secantelor este un algoritm iterativ în doi paşi (aproximația x_{k+1} se calculează în funcție de două aproximații anterioare x_k şi x_{k-1}) şi, în condiții identice cu cele impuse asupra funcției $f(x)$ în metoda lui Newton, converge monoton şi unilateral. Viteza de convergență a metodei în vecinătatea rădăcinii simple x^* este superliniară şi se determină de relația [17; 21; 23-24]:

$$\left|x_{k+1}-x^*\right| \le C\left|x_k-x^*\right|^m,$$

în care indicele este $m = 0.5\cdot\left(1+\sqrt{5}\right) \approx 1.618$.

Numărul minim de iterații n, necesare pentru calculul rădăcinii cu exactitatea relativă ε este $n = [1.44\cdot \log_2 \log_2 1/\varepsilon] + 1$ [17]. Deşi metoda secantelor converge mai lent decât metoda Newton, efectuând acelaşi volum de operații, prin metoda secantelor s-ar putea obține o precizie mai mare, ținând cont că aici se cere numai calculul valorilor funcției, dar nu şi al valorilor derivatei.

În numitorul formulei de calcul (4.5) avem diferența valorilor funcției, ceea ce poate conduce la pierderea cifrelor semnificative în vecinătatea rădăcinii, deoarece valorile funcției devin mici şi aproximativ egale. Apare o acumulare esențială a erorii în urma căreia suntem mărginiți în calculul rădăcinii cu o precizie înaltă. Problema dată poate fi depăşită astfel. Se alege un număr ε_1 nu prea mic şi se fac iterații până atunci când se va îndeplini inegalitatea $|x_{k+1}-x_k| < \varepsilon_1$. Apoi calculele continuă, cât mai descreşte mărimea $|x_{k+1}-x_k|$. Prima creştere a mărimii $|x_{k+1}-x_k|$ înseamnă începutul manifestării fenomenului descris şi, de aceea, iterațiile se întrerup. Algoritmul descris este cunoscut ca procedeul lui Garvik [16-17; 24]. Pentru a depăşi problema menționată, deseori se aplică mai întâi metoda secantelor, apoi se trece la aplicarea metodei lui Newton.

♦ **Metoda coardelor (interpolării liniare, falsei poziții, secantelor cu un capăt fixat)**

În formula de calcul (4.5) se fixează aproximația $x_{k-1} = x_0$ pentru toți $k = 2,3,...$, apoi, în baza aceleiaşi formule, se calculează aproximația x_{k+1} $(k = 1,2,...)$ astfel, încât rădăcina ecuației să aparțină intervalului $[x_{k+1}; x_0]$ (dacă $x_{k+1} < x_0$) sau $[x_0; x_{k+1}]$ (dacă $x_0 < x_{k+1}$). Astfel, în condiții identice cu cele impuse asupra funcției $f(x)$ în metoda lui Newton, dacă $f(a_0)f''(a_0) > 0$ se consideră în formula (4.5) $x_{k-1} = a_0$ $(k = 1,2,...)$, iar în cazul în care $f(a_0)f''(a_0) < 0$, avem $x_{k-1} = b_0$ $(k = 1,2,...)$.

Aproximația x_{k+1} a rădăcinii se determină ca punctul de intersecție cu axa absciselor al coardei care trece prin punctele de coordonate $(x_k; f(x_k))$ $(k \ge 1)$ şi $(x_0; f(x_0))$. Dintre intervalele formate se selectează acela în extremitățile căruia funcția are valori de semne contrare.

Prin urmare, formula de calcul a metodei coardelor se defineşte prin următoarea relație de recurență:

$$x_{k+1}=\begin{cases} x_k-\dfrac{f(x_k)}{f(x_k)-f(a_0)}(x_k-a_0) & (k=1,2,...)\ \ dacă\ \ f(a_0)f''(a_0)>0 \\ x_k-\dfrac{f(x_k)}{f(x_k)-f(b_0)}(x_k-b_0) & (k=1,2,...)\ \ dacă\ \ f(a_0)f''(a_0)<0 \end{cases}.$$

Aproximația inițială $x_1\in[a_0;b_0]$ se alege în modul următor:

$$x_1=\begin{cases} b_0 & dacă\ \ f(a_0)f''(a_0)>0 \\ a_0 & dacă\ \ f(a_0)f''(a_0)<0 \end{cases}.$$

Drept criteriu de stopare a calculelor în procesul iterativ serveşte inegalitatea [2-4; 16; 23-24]:

$$\left|x_{k+1}-x^*\right|\le\frac{M_1-m_1}{m_1}\left|x_{k+1}-x_k\right|<\varepsilon\ \ (k=1,2,...);\ M_1=\max_{x\in[a_0;b_0]}\left|f'(x)\right|,\quad m_1=\min_{x\in[a_0;b_0]}\left|f'(x)\right|. \tag{4.6}$$

În cazul în care calculul constantelor M_1 şi m_1 întâmpină dificultăți, pentru a calcula rădăcina ecuației (4.1) cu exactitatea ε se poate utiliza condiția ca mărimile $\left|x_{k+1}-x_k\right|$ şi $\left|f(x_{k+1})\right|$ să devină mai mici ca ε simultan.

Pseudocodul algoritmului

Date de intrare:
- f (f'') – numele subprogramului-funcție (de tip real), cu ajutorul căruia se calculează valoarea funcției $f(x)$ (derivatei a doua $f''(x)$);
- a,b (de tip real) – extremitățile segmentului pe care este separată rădăcina;
- ε (de tip real) – precizia dorită;

Date de ieşire:
- x (de tip real) – aproximația rădăcinii ecuației;
- fx (de tip real) – valoarea funcției f în punctul x;
- k (de tip întreg) – numărul de iterații efectuate;

```
întreg k
real x, x_f, d, fx, fx_f
dacă f(a)f''(a)>0 atunci {x:=b; x_f:=a}
altfel {x:=a; x_f:=b}
k:=0; fx:=f(x); fx_f:=f(x_f)
execută
  {
    d:=(x-x_f)fx/(fx-fx_f); x:=x-d; fx:=f(x); k:=k+1
  }
cât timp (|d|≥ε sau |fx|≥ε)
scrie x, fx, k
```

Analiza algoritmului

Metoda coardelor este un algoritm iterativ într-un singur pas şi, în condiții identice cu cele impuse funcției $f(x)$ în metoda lui Newton, converge monoton şi unilateral. Viteza de convergență a metodei în vecinătatea rădăcinii simple x^* este liniară (a se vedea relația (4.6)).

Analiza formulei de calcul a metodei coardelor (dar şi a formulei metodei secantelor) ne sugerează că aceasta poate fi aplicată în calculul rădăcinilor ecuației (4.1), atunci când funcția $f(x)$ este continuă pe $[a_0;b_0]$. Astfel, metoda coardelor (ca şi metoda secantelor), spre deosebire de metoda lui Newton, este

aplicabilă unei clase mai largi de funcții $f(x)$. Totodată, vom ține cont că existența derivatelor $f'(x)$ și $f''(x)$, care nu se anulează pe intervalul de localizare $[a_0;b_0]$, asigură aplicarea cu succes a metodei.

♦ **Metoda iterației simple (metoda aproximațiilor succesive, metoda contracțiilor)**

Aplicarea metodei necesită înlocuirea ecuației (4.1) cu o ecuație echivalentă de forma:

$$x=\varphi(x), \tag{4.7}$$

unde $\varphi(x)\in C^1[a_0;b_0]$. Se alege aproximația inițială $x_0\in[a_0;b_0]$ și se generează succesiv șirul aproximațiilor conform formulei

$$x_{k+1}=\varphi(x_k),\ k=0,1,\ldots. \tag{4.8}$$

Dacă șirul $\{x_k\}_{k\geq 0}$ este convergent, atunci limita acestuia este rădăcina ecuației (4.1) de pe $[a_0;b_0]$.

Există mai multe posibilități de a reprezenta în mod echivalent ecuația (4.1) sub forma (4.7), reprezentări ce asigură convergența procesului iterativ (4.8) atunci, când aproximația inițială x_0 se ia dintr-o vecinătate suficient de mică a rădăcinii. În continuare, vom descrie un astfel de procedeu de construcție a funcției $\varphi(x)$, care asigură convergența metodei iterației simple pentru orice aproximație inițială $x_0\in[a_0;b_0]$. Ipotezele necesare sunt următoarele: $x^*\in[a_0;b_0]\subset[a;b]$ este rădăcina separată pe intervalul indicat, $f(x)\in C^1[a_0;b_0]$, iar $f'(x)$ păstrează semnul constant pe $[a_0;b_0]$. În cazul în care $f'(x)>0,\ \forall x\in[a_0;b_0]$, funcția $\varphi(x)$ se ia de forma $\varphi(x)=x-\lambda f(x)$, iar atunci când $f'(x)<0$, $\forall x\in[a_0;b_0]$, de forma $\varphi(x)=x+\lambda f(x)$. Valoarea parametrului λ, în ambele cazuri, se determină de una dintre formulele $\lambda\equiv\lambda^{(1)}=2/(M+m)$ sau $\lambda\equiv\lambda^{(2)}=1/M$, în care $m=\min\limits_{x\in[a_0;b_0]}|f'(x)|$, $M=\max\limits_{x\in[a_0;b_0]}|f'(x)|$ [3; 8; 16; 21].

Pentru a calcula rădăcina x^* cu exactitatea cerută ε, se utilizează una dintre relațiile [2-4; 8; 16-17]:

$$\begin{aligned}&|x_{k+1}-x^*|\leq\frac{q}{1-q}|x_{k+1}-x_k|,\ k=0,1,\ldots,\\ &|x_{k+1}-x^*|\leq\frac{q^{k+1}}{1-q}|x_1-x_0|,\ k=0,1,\ldots.\end{aligned} \tag{4.9}$$

Constanta q se calculează astfel: $q\equiv q^{(1)}=(M-m)/(M+m)$ sau $q\equiv q^{(2)}=1-m/M$ [16-17].

Pseudocodul algoritmului

Date de intrare: f (f') – numele subprogramului-funcție (de tip real), cu ajutorul căruia se calculează valoarea funcției $f(x)$ (derivatei $f'(x)$); a (de tip real) – extremitatea de stânga a segmentului de localizare a rădăcinii; m,M (de tip real) – constante ce figurează în formula (4.9) pentru stoparea calculelor; ε (de tip real) – precizia dorită. Date de ieşire: x (de tip real) – aproximația rădăcinii ecuației; $f(x)$ (de tip real) – valoarea funcției f în punctul x; k (de tip întreg) – numărul de iterații efectuate; întreg k real $x,q,\varepsilon_1,\lambda,d$
$q:=(M-m)/(M+m);\ \varepsilon_1:=(1-q)\varepsilon/q;\ \lambda:=2/(M+m);\ k:=0;\ x:=a$ dacă $f'(x)<0$ atunci $\lambda:=-\lambda$ execută

```
    {
      d := λ f(x); x := x − d; k := k + 1
    }
  cât timp |d| ≥ ε1
scrie x, f(x), k
```

Analiza algoritmului

Dacă pe $[a_0;b_0]$ se îndeplineşte inegalitatea $0<\varphi'(x)<1$, atunci şirul de aproximaţii generat conform metodei iteraţiei simple converge monoton şi unilateral către rădăcină [2-5; 8; 10; 13; 15-17]. Dacă, însă $-1<\varphi'(x)<0$, atunci convergenţa este bilaterală [2-5; 8; 15-17], caz în care se poate lua drept criteriu de stopare a calculelor îndeplinirea condiţiei $|x_{k+1}-x_k|<\varepsilon$. Dacă $|\varphi'(x)|<1$ pe o careva vecinătate a rădăcinii, iar în exteriorul acesteia avem $|\varphi'(x)|>1$, atunci iteraţiile vor converge când aproximaţia iniţială x_0 este aleasă suficient de aproape de rădăcină. În acest caz, pentru o aproximaţie iniţială arbitrară s-ar putea să nu avem convergenţă.

Numărul minim de iteraţii n necesare pentru a calcula rădăcina cu exactitatea ε este $n=\left[\ln\left(|x_1-x_0|/(1-q)\varepsilon\right)/\ln(1/q)\right]+1$ [17]. În general, viteza de convergenţă a procesului iterativ este liniară (a se vedea relaţia (4.9)).

♦ **Calculul rădăcinilor ecuaţiilor polinomiale**

Metodele numerice, analizate mai sus, se aplică cu succes şi în calculul rădăcinilor ecuaţiilor polinomiale $f(x)=0$, unde $f(x)\equiv p(x)=\alpha_n x^n+\alpha_{n-1}x^{n-1}+...+\alpha_1 x+\alpha_0$. Vom analiza, totuşi, unele aspecte ce ţin de implementarea metodei lui Newton. Conform formulei de calcul (4.2), realizarea iteraţiei cu numărul $k+1$ necesită calculul valorilor $p(x_k)$ şi $p'(x_k)$. Deoarece $p(x)$ şi $p'(x)$ sunt polinoame, valorile acestora pot fi calculate eficient, cu un număr minim de operaţii aritmetice, utilizând algoritmul lui Horner [24-25]. Conform acestuia, dacă $\beta_n=\alpha_n$ şi $\beta_j=\alpha_j+\beta_{j+1}x_k$, $j=n-1,n-2,...,0$, atunci $p(x_k)=\beta_0$. Mai mult, dacă $q(x)=\beta_n x^{n-1}+\beta_{n-1}x^{n-2}+...+\beta_2 x+\beta_1$, atunci $p(x)=(x-x_k)q(x)+\beta_0$. Derivând ambii membri ai ultimei relaţii, obţinem $p'(x)=q(x)+(x-x_k)q'(x)$, de unde $p'(x_k)=q(x_k)$. Valoarea $q(x_k)$ se calculează utilizând formulele: $\gamma_{n-1}=\alpha_n$, $\gamma_{j-1}=\beta_j+\gamma_j x_k$, $j=n-1,n-2,...,1$, $q(x_k)=\gamma_0$. Prin urmare, atunci când se aplică metoda lui Newton în calculul rădăcinilor ecuaţiilor polinomiale, utilizarea algoritmului lui Horner permite calculul simultan al valorilor $p(x_k)$ şi $p'(x_k)$.

Pseudocodul algoritmului

Date de intrare:
- n (de tip întreg) – gradul polinomului $p(x)$;
- $\alpha[n]$ – tabloul (cu elemente de tip real; indici de la 0 la n) coeficienţilor polinomului $p(x)$;
- a,b (de tip real) – extremităţile segmentului pe care este separată rădăcina;
- ε (de tip real) – precizia dorită;
- m_1,M_2 (de tip real) – constante ce figurează în formula (4.4) pentru stoparea calculelor.

Date de ieşire:
- x (de tip real) – aproximaţia rădăcinii ecuaţiei;
- $f(x)$ (de tip real) – valoarea funcţiei f în punctul x;
- k (de tip întreg) – numărul de iteraţii efectuate;

întreg k
real $x, y, z, \varepsilon_1, d$

$\varepsilon_1 := \sqrt{2m_1\varepsilon / M_2}$; $k := 0$
dacă $f(a)f''(a) > 0$ atunci $x := a$
altfel $x := b$
execută
{
$y := \alpha_n$; $z := \alpha_n$
pentru $j := \overline{n-1,1}$ execută { $y := xy + \alpha_j$; $z := xz + y$ }
$y := xy + \alpha_0$; $d := y/z$; $x := x - d$; $k := k + 1$
}
cât timp $|d| \geq \varepsilon_1$
scrie $x, f(x), k$

♦ **Calculul rădăcinilor multiple**

Fie $x^* \in [a_0; b_0]$ o rădăcină a ecuației (4.1) de multiplicitate $m > 1$ ($m \in \mathbb{N}$), iar $f \in C^2[a_0; b_0]$. În acest caz, oricât de îngust nu ar fi intervalul $[a_0; b_0]$ ce îl conține pe x^*, funcția $f(x)$ nu va satisface pe acesta toate condițiile ce asigură convergența proceselor iterative analizate mai sus. Şi chiar dacă, în anumite condiții, metoda lui Newton sau metoda secantelor converge, viteza de convergență a acestora este mai mică decât în cazul când se aplică în calculul unei rădăcini simple a ecuației (4.1). De exemplu, în condițiile impuse mai sus funcției $f(x)$, în vecinătatea rădăcinii, metoda lui Newton converge cu viteza liniară a unei progresii geometrice cu rația $(m-1)/m$. De aceea, algoritmii metodelor iterative pentru calculul rădăcinilor simple ale ecuației (4.1), se modifică astfel, încât să conveargă garantat şi să păstreze rata convergenței atunci, când sunt utilizați în calculul rădăcinilor multiple. Rădăcina $x^* \in [a_0; b_0]$ a ecuației (4.1), de multiplicitate m, este şi rădăcină a ecuației $f'(x) = 0$, dar deja de multiplicitate $m-1$. Atunci x^* este rădăcină simplă a ecuației $g(x) = 0$, unde $g(x) = f(x)/f'(x)$. Când funcția $g(x)$ este suficient de netedă, rădăcina x^* poate fi calculată, utilizând algoritmii iterativi analizați mai sus.

4.III. Modul de desfăşurare a lucrării

♦ **Cerințe față de îndeplinirea lucrării**

Să se elaboreze un program, destinat localizării tuturor rădăcinilor reale ale ecuației $f(x) = 0$ pe intervalul specificat $[a; b]$. Să se implementeze algoritmii metodelor numerice (analizate mai sus) pentru calculul rădăcinilor ecuației $f(x) = 0$ într-un limbaj de programare de nivel înalt. Programul de calcul să se testeze pe exemplele propuse la sfârşitul lucrării de laborator.

Să se extragă la tipar rădăcinile ecuației, valorile funcției $f(x)$ pentru rădăcini, numărul de iterații, necesar pentru a obține exactitatea cerută $\varepsilon > 0$, precum şi timpii necesari pentru efectuarea unei iterații şi pentru atingerea unei precizii dorite. În plus, pentru fiecare metodă se va afişa şi reprezenta grafic variația erorii aparente Cauchy $|x_{k+1} - x_k|$ (distanța dintre două iterații succesive) şi a reziduului $|f(x_{k+1})|$ pe parcursul iterațiilor. Se vor compara rezultatele obținute prin diferite metode (eficiența lor relativă în rezolvarea problemei date) şi se va determina metoda optimă.

♦ **Chestiuni de studiat**

1. Comparați eficiența diferitelor metode iterative pentru rezolvarea aceleiaşi ecuații.
2. Analizați cum depinde eficiența rezolvării de aproximația inițială adoptată.
3. Să se facă o analiză experimentală a dependenței timpului de calcul al rădăcinii ecuației printr-o metodă iterativă de exactitatea de calcul impusă.
4. Propuneți diferite criterii sigure pentru stoparea iterațiilor.
5. Pentru fiecare metodă iterativă analizată, să se formuleze un exemplu de ecuație, astfel încât respectivul proces iterativ să nu conveargă.

♦ **Probleme propuse spre rezolvare**

1*a*. $\ln(x/4)+4\sin 3x+1=0;\ [a;b]=[0.1;5];$	14*a*. $\ln x-\cos x-\sqrt{x}=0;\ [a;b]=[1;11];$
2*a*. $\ln(8.5x)-9.6\cos 2x=0;\ [a;b]=[0.5;9];$	15*a*. $6x\sin x^2+x^3-1=0;\ [a;b]=[-3;3];$
3*a*. $e^{-x}+x^3-1=0;\ [a;b]=[-5;2];$	16*a*. $e^{x^2}-x-4\cos 6x-1=0;\ [a;b]=[-3;2];$
4*a*. $2^x+x^2-6\sin 2x=0;\ [a;b]=[-3;2];$	17*a*. $\cos x+x^3-e^x=0;\ [a;b]=[-2;6];$
5*a*. $x^2-5\sin 2x-0.3=0;\ [a;b]=[-3;2];$	18*a*. $x^3-5arctgx=0;\ [a;b]=[-2;3];$
6*a*. $\ln x-(x-2)^3-5\sin 3x=0;\ [a;b]=[0.1;4];$	19*a*. $-\ln x-3\cos 2x=0;\ [a;b]=[0.5;8];$
7*a*. $2\sin 3x-\ln(x^3-1)+4=0;\ [a;b]=[2;9];$	20*a*. $e^x-3\cos 2x=0;\ [a;b]=[-9;2];$
8*a*. $\sin(\pi x/6)-\cos(x-1)=0;\ [a;b]=[-7;8];$	21*a*. $e^{2x}-\cos 3x=0;\ [a;b]=[-9;1];$
9*a*. $e^x-x^2-4x-3=0;\ [a;b]=[-4;4];$	22*a*. $shx-5\cos 3x=0;\ [a;b]=[-2;3];$
10*a*. $10^x+\sin x=0;\ [a;b]=[-10;1];$	23*a*. $e^{-2x}-\sin x=0;\ [a;b]=[0;10];$
11*a*. $e^{-x}-x^3+8\cos 4x=0;\ [a;b]=[-4;4];$	24*a*. $e^{-3x}-\cos x=0;\ [a;b]=[-1;9];$
12*a*. $chx-3\sin 3x=0;\ [a;b]=[-2;3];$	25*a*. $arctgx+2\sin x^2=0;\ [a;b]=[-2;3];$
13*a*. $x\cos x-\sin x=0;\ [a;b]=[-9;9];$	26*a*. $\sin x^2+2\cos x=0;\ [a;b]=[-6;6];$

1*b*. $x^6-5.5x^5+6.18x^4+18.54x^3-56.9592x^2+55.9872x-19.3156=0;\ [a;b]=[-3;4];$

2*b*. $x^6-0.7x^5-8.7x^4+5.58x^3+22.356x^2-8.39808x-19.3156=0;\ [a;b]=[-3;4];$

3*b*. $x^6-2.4x^5-18.27x^4+23.216x^3+115.7x^2-19.5804x-164.818=0;\ [a;b]=[-3;4];$

4*b*. $x^6-1.8x^5-18.87x^4+17.06x^3+117.533x^2-14.2296x-182.613=0;\ [a;b]=[-3;4];$

5*b*. $x^6-1.6x^5-20.46x^4+14.432x^3+134.548x^2+4.18968x-204.99=0;\ [a;b]=[-3;4];$

6*b*. $x^6-1.4x^5-22.07x^4+11.292x^3+152.842x^2+28.1318x-227.101=0;\ [a;b]=[-3;4];$

7*b*. $x^6-1.2x^5-25.35x^4+8.45x^3+197.73x^2+61.6918x-320.797=0;\ [a;b]=[-3;4];$

8*b*. $x^6-0.8x^5-24.23x^4+2.786x^3+178.428x^2+63.5305x-288.739=0;\ [a;b]=[-3;4];$

9*b*. $x^6-0.6x^5-23.64x^4+0.152x^3+169.104x^2+64.247x-273.342=0;\ [a;b]=[-3;4];$

10*b*. $x^6-0.4x^5-23.03x^4-2.35x^3+159.999x^2+64.8284x-258.367=0;\ [a;b]=[-3;4];$

11*b*. $x^6-0.2x^5-22.4x^4-4.72x^3+151.112x^2+65.2746x-243.814=0;\ [a;b]=[-3;4];$

12*b*. $x^6+0.2x^5-21.08x^4-9.064x^3+133.994x^2+65.7613x-215.974=0;\ [a;b]=[-3;4];$

13*b*. $x^6+0.4x^5-20.39x^4-11.038x^3+125.762x^2+65.8018x-202.686=0;\ [a;b]=[-3;4];$

14*b*. $x^6+0.7x^5-21.02x^4-15.582x^3+133.82x^2+83.3903x-226.984=0;\ [a;b]=[-3;4];$

15*b*. $x^6+0.5x^5-21.77x^4-13.665x^3+142.787x^2+83.9808x-241.865=0;\ [a;b]=[-3;4];$

16*b*. $x^6+0.3x^5-22.5x^4-11.61x^3+151.997x^2+84.4401x-257.217=0;\ [a;b]=[-3;4];$

17*b*. $x^6+0.1x^5-23.21x^4-9.417x^3+161.449x^2+84.7681x-273.043=0;\ [a;b]=[-3;4];$

18*b*. $x^6-0.1x^5-23.9x^4-7.086x^3+171.145x^2+84.9649x-289.34=0;\ [a;b]=[-3;4];$

19*b*. $x^6-0.3x^5-24.57x^4-4.617x^3+181.084x^2+85.0306x-306.11=0;\ [a;b]=[-3;4];$

20*b*. $x^6-0.5x^5-25.22x^4-2.01x^3+191.265x^2+84.9649x-323.352=0;\ [a;b]=[-3;4];$

21*b*. $x^6-0.7x^5-25.85x^4+0.735x^3+201.69x^2+84.7681x-341.067=0;\ [a;b]=[-3;4];$

22*b*. $x^6-0.9x^5-26.46x^4+3.618x^3+212.358x^2+84.4401x-359.254=0;\ [a;b]=[-3;4];$

23*b*. $x^6-0.6x^5-27.51x^4-1.592x^3+227.002x^2+110.074x-400.668=0;\ [a;b]=[-3;4];$

24*b*. $x^6-0.4x^5-26.84x^4-4.448x^3+215.398x^2+109.635x-380.384=0;\ [a;b]=[-3;4];$

25*b*. $x^6+0.8x^5-22.4x^4-18.56x^3+151.424x^2+104.366x-269.746=0;\ [a;b]=[-3;4];$

26*b*. $x^6+1.3x^5-22.1x^4-25.51x^3+149.365x^2+124.88x-281.254=0;\ [a;b]=[-3;4].$

LUCRAREA DE LABORATOR №5
REZOLVAREA NUMERICĂ A SISTEMELOR DE ECUAȚII ALGEBRICE LINIARE

I. Scopul lucrării

Studiul unor metode numerice de rezolvare a sistemelor de ecuații algebrice liniare şi implementarea acestora într-un limbaj de programare de nivel înalt.

II. Prezentarea teoretică a lucrării

♦ Formularea problemei

Să considerăm sistemul de ecuații algebrice liniare (prescurtat SEAL):

$$A\bar{x} = \bar{b}\,, \tag{5.1}$$

în care A este o matrice pătrată de ordinul n cu elementele numere reale, iar $\bar{x}$ şi $\bar{b}$ sunt vectori cu componente din spațiul euclidian n- dimensional $\mathbb{R}^n$. Cunoscând matricea $A=\{a_{ij}\}_{i,j=1}^{n}$ a coeficienților sistemului şi vectorul $\bar{b}=\{b_i\}_{i=1}^{n}$ al termenilor liberi, se pune problema determinării soluției $\bar{x}=\{x_j\}_{j=1}^{n}$. În continuare, vom presupune că $\det A\neq 0$ (matricea A este nesingulară), condiție ce asigură existența şi unicitatea soluției SEAL (5.1).

Metodele de rezolvare a SEAL le putem împărți în două categorii:

a) Metode directe (sau metode exacte) ce se caracterizează prin faptul că soluția sistemului se obține printr-o secvență de operații care se execută o singură dată, numărul total de operații aritmetice elementare fiind finit şi cunoscut de la început. Soluția calculată printr-o metodă directă diferă de soluția exactă datorită erorilor de rotunjire.

b) Metode iterative conform cărora soluția sistemului se obține printr-un proces de aproximații succesive. O secvență de operații este parcursă de mai multe ori, obținând aproximații tot mai bune ale soluției, până la atingerea unei precizii fixate în prealabil.

În cele ce urmează vom prezenta cele mai cunoscute metode de rezolvare a SEAL (5.1) (reprezentante ale ambelor categorii menționate) şi vom face o analiză comparativă a eficienței acestora.

♦ Metode directe

Metoda eliminării a lui Gauss constă din două etape [1-5; 10-11; 15-16; 21-26]:

1) transformarea sistemului (5.1) într-un sistem echivalent cu matrice superior triunghiulară:

$$\sum_{j=i}^{n} a_{ij}^{(i)} x_j = b_i^{(i)},\ i=\overline{1,n}; \tag{5.2}$$

2) rezolvarea sistemului (5.2) utilizând metoda substituției inverse (eliminării inverse).

Vom utiliza notațiile $b_i^{(1)} := b_i,\ a_{ij}^{(1)} := a_{ij},\ i,j=\overline{1,n}$. Soluția sistemului (5.1) se obține prin aplicarea formulelor:

$$m_{ik} = a_{ik}^{(k)} \big/ a_{kk}^{(k)},\ b_i^{(k+1)} = b_i^{(k)} - m_{ik} b_k^{(k)},\ a_{ij}^{(k+1)} = a_{ij}^{(k)} - m_{ik} a_{kj}^{(k)},\ i,j=\overline{k+1,n}, \tag{5.3}$$

pentru $k=\overline{1,n-1}$, apoi a formulelor:

$$x_n = b_n^{(n)} \big/ a_{nn}^{(n)},\ x_k = \left(b_k^{(k)} - \sum_{j=k+1}^{n} a_{kj}^{(k)} x_j \right) \Big/ a_{kk}^{(k)},\ k=\overline{n-1,1}, \tag{5.4}$$

în ipoteza că $a_{kk}^{(k)} \neq 0,\ k=\overline{1,n}$.

Se poate utiliza şi o formă modificată a metodei de eliminare a lui Gauss, cunoscută sub numele de ***algoritmul de eliminare a lui Gauss-Jordan*** [3; 10-11; 15-16; 21-26], conform căruia sistemul (5.1) se transformă într-un sistem echivalent cu matrice diagonală. În acest caz se simplifică etapa a doua, în care se calculează soluția sistemului. Algoritmul de calcul, care conduce la sistemul cu matrice diagonală, este definit de formulele (5.3) şi (5.4) în care indicii i, j şi k variază în limitele următoare: $k=\overline{1,n},\ i=1,...,k-1,k+1,...,n,\ j=\overline{k+1,n}$. Componentele soluției se determină prin formula $x_k = b_k^{(k)} / a_{kk}^{(k)}$, $k=\overline{1,n}$.

Pseudocodul algoritmului

Date de intrare:
n (de tip întreg) – ordinul sistemului liniar;
$a[n,n]$ - tablou bidimensional (cu elemente de tip real; indici de la 1 la n) ce conține componentele matricei coeficienților sistemului;
$b[n]$ - tablou (cu elemente de tip real; indici de la 1 la n) ce conține componentele vectorului termenilor liberi ai sistemului.
Date de ieşire:
$x[n]$ - tablou (cu elemente de tip real; indici de la 1 la n) ce conține componentele soluției sistemului.

Algoritmul de eliminare a lui Gauss (vom nota a.e.G)

```
întreg i, j, k
real s, m
tablou real x[n]
 pentru k := 1,n-1 execută
  {
 * pentru i := k+1,n execută
    {
     dacă a_ik ≠ 0 atunci
      {
        m := a_ik / a_kk ; a_ik := 0; b_i := b_i - m b_k
        pentru j := k+1,n execută a_ij := a_ij - m a_kj
      }
    }
  }
 dacă a_nn = 0 atunci { se afişează mesajul
 "matricea sistemului este singulară (MSS)"; STOP }
 x_n := b_n / a_nn
 pentru k := n-1,1 execută
  {
** s := 0
   pentru j := k+1,n execută s := s + a_kj x_j
   x_k := (b_k - s) / a_kk
  }
pentru k := 1,n scrie x_k
```

Algoritmul de eliminare a lui Gauss-Jordan (vom nota a.e.GJ)

```
întreg i, j, k
real m
tablou real x[n]
 pentru k := 1,n execută
  {
 * pentru i := 1,n execută
    {
     dacă a_ik ≠ 0 atunci
      {
       dacă i ≠ k atunci
        {
          m := a_ik / a_kk ; a_ik := 0; b_i := b_i - m b_k
          pentru j := k+1,n execută a_ij := a_ij - m a_kj
        }
      }
    }
  }
 dacă a_nn = 0 atunci {se afişează mesajul "matricea
** sistemului este singulară (MSS)"; STOP }
 pentru k := 1,n execută x_k := b_k / a_kk
pentru k := 1,n scrie x_k
```

Remarcă. Pentru soluția $\bar{x}$ nu este necesară rezervarea unei zone de memorie specială, în acest sens, memoria alocată vectorului termenilor liberi poate fi destinată şi soluției. În plus, întrucât la etapa k a algoritmilor menționați se anulează elementele $a_{ik}^{(k)},\ i=\overline{k+1,n}$, locațiile de memorie rezervate pentru

acestea pot fi utilizate pentru a memora multiplicatorii m_{ik}.

Strategii de pivotare în rezolvarea SEAL

Utilizarea a.e.G sau a.e.GJ la rezolvarea sistemului (5.1) poate conduce la situația în care un element diagonal $a_{kk}^{(k)}$ este nul, chiar dacă matricea sistemului este nesingulară. Pentru a evita o împărțire la zero, este necesară efectuarea operației de pivotare, conform căreia se permută liniile şi/sau coloanele matricei coeficienților astfel, încât să se aducă pe poziția diagonală (k,k) un element nenul. Utilizarea operației de pivotare permite nu doar extinderea domeniului de aplicabilitate al algoritmilor analizați, ci şi îmbunătățirea stabilității numerice a acestora. Pentru a asigura o bună stabilitate numerică (erori de rotunjire minime) a procesului de calcul se preferă alegerea în calitate de pivot a unui element de modul maxim. Se pot utiliza două strategii de pivotare: totală şi parțială [1-5; 8; 10-11; 15-16; 21-26].

Pivotarea totală constă în căutarea pivotului de modul maxim printre toți coeficienții $a_{ij}^{(k)}, i, j = \overline{k,n}$, (aflați sub linia k şi la dreapta coloanei k) şi aducerea acestui element pe poziția (k,k) prin permutări de linii şi de coloane în matricea A şi permutări corespunzătoare în vectorul termenilor liberi (se modifică atât ordinea ecuațiilor, cât şi ordinea componentelor soluției în sistem). Astfel, această strategie impune memorarea permutărilor de coloane făcute pe parcursul algoritmului, pentru a se putea reconstitui la sfârşit ordinea inițială a necunoscutelor.

Pentru a evita unele permutări de coloane, se poate folosi o *pivotare parțială*, care constă în căutarea pivotului de modul maxim printre toți coeficienții $a_{ik}^{(k)}, i = \overline{k,n}$ (aflați sub linia k şi pe coloana k) şi aducerea acestui element pe poziția (k,k) prin permutări de linii în matricea A şi permutări corespunzătoare în vectorul termenilor liberi (ceea ce este echivalent cu schimbarea ordinii ecuațiilor). Totuşi, în cazul în care $a_{kk}^{(k)} \approx 0$, iar $a_{ik}^{(k)} = 0, i = \overline{k+1,n}$, pentru a nu introduce o eroare semnificativă în calcule, se va face uz şi de permutarea coloanelor.

Pseudocodul algoritmului

Date de intrare:
n (de tip întreg) – ordinul sistemului liniar;
$a[n,n]$ – tablou bidimensional (cu elemente de tip real; indici de la 1 la n) ce conține componentele matricei coeficienților sistemului;
$b[n]$ - tablou (cu elemente de tip real; indici de la 1 la n) ce conține componentele vectorului termenilor liberi ai sistemului.
Date de ieşire:
$b[n]$ - tablou (cu elemente de tip real; indici de la 1 la n) ce conține componentele soluției sistemului.

Algoritmul de eliminare cu pivotare totală

```
întreg i, i_max, j, j_max, k
real s, m, max
tablou real x[n]
tablou întreg p[n]
 pentru j := 1,n execută p_j := j
 pentru k := 1,n-1 execută
  {
    max := 0.0; i_max := k; j_max := k
    pentru i := k,n execută
```

Algoritmul de eliminare cu pivotare parțială

```
întreg i, j, k, max
real s, m
 pentru k := 1,n-1 execută
  {
    max := k
    pentru j := k+1,n execută
      { dacă |a_jk| > |a_max,k| atunci max := j }
    dacă a_max,k = 0 atunci {se afişează mesajul "MSS";
        STOP }
```

```
    {
      pentru j := \overline{k,n} execută
       {
         dacă |a_ij| > max atunci
           { max := |a_ij|; i_max := i; j_max := j }
       }
    }
    dacă max = 0 atunci
       {se afişează mesajul "MSS"; STOP }
    dacă i_max ≠ k atunci
     {
       pentru j := \overline{k,n} execută
         { m := a_kj; a_kj := a_{i_max,j}; a_{i_max,j} := m }
       m := b_k; b_k := b_{i_max}; b_{i_max} := m
     }
    dacă j_max ≠ k atunci
     {
       pentru i := \overline{k,n} execută
         { m := a_ik; a_ik := a_{i,j_max}; a_{i,j_max} := m }
       j := p_k; p_k := p_{j_max}; p_{j_max} := j
     }
    aici urmează secvenţa (*) a unuia dintre
    algoritmii de la pagina 47 (pentru a.e.GJ avem
    k = \overline{1,n})
  }
  aici urmează secvenţa (**) a unuia dintre
  algoritmii de la pagina 47
  pentru k := \overline{1,n} execută b_{p_k} := x_k
pentru k := \overline{1,n} scrie b_k
```

```
    dacă max ≠ k atunci
     {
       pentru j := \overline{k,n} execută
        { m := a_kj; a_kj := a_{max,j}; a_{max,j} := m }
       m := b_k; b_k := b_max; b_max := m
     }
   aici urmează secvenţa (*) a unuia dintre algoritmii
   de la pagina 47 (pentru a.e.GJ avem k = \overline{1,n})
  }
  aici urmează secvenţa (**) a unuia dintre algoritmii
  de la pagina 47 (se va ţine cont de remarca de la
  pagina 47)
pentru k := \overline{1,n} scrie x_k
```

Aplicații ale algoritmilor de eliminare

A) *Rezolvarea SEAL cu matrice rare*

O aplicație importantă a metodei de eliminare a lui Gauss ține de rezolvarea SEAL cu matrice de formă specială. Vom considera cazul în care matricea coeficienților A a sistemului (5.1) este tridiagonală, adică are loc egalitatea $a_{ij}=0$ pentru toți i, j $(i, j=\overline{1,n})$ ce satisfac relația $|i-j|>1$. Pentru rezolvarea a astfel de sisteme se aplică cu succes **metoda parcurgerii** [10-11; 16; 22-23; 27], care reprezintă o adaptare a algoritmului de eliminare a lui Gauss.

Fie $\overline{c}=\{c_i\}_{i=2}^{n}$ - codiagonala inferioară a matricei A, $\overline{d}=\{d_i\}_{i=1}^{n}$ - diagonala principală, iar $\overline{e}=\{e_i\}_{i=1}^{n-1}$ - codiagonala superioară. *Metoda parcurgerii de dreapta* definește următorul proces de calcul al soluției sistemului (5.1) cu matrice tridiagonală:

$$\begin{gathered}\alpha_1=-e_1/d_1,\ \beta_1=b_1/d_1,\ h_i=d_i+c_i\alpha_{i-1},\ \alpha_i=-e_i/h_i,\ \beta_i=\left(b_i-c_i\beta_{i-1}\right)/h_i,\ i=\overline{2,n-1},\\ x_n=\left(b_n-c_n\beta_{n-1}\right)/\left(c_n\alpha_{n-1}+d_n\right),\ x_i=\alpha_i x_{i+1}+\beta_i,\ i=\overline{n-1,1}.\end{gathered} \tag{5.5}$$

Realizarea algoritmului este posibilă dacă $d_1\neq 0,\ c_n\alpha_{n-1}+d_n\neq 0,\ h_i\neq 0,\ i=\overline{2,n-1}$, iar ultimele relații se vor satisface dacă matricea A este pozitiv definită sau este slab diagonal dominantă, adică dacă se satisface

relația $|d_i| \geq |c_i| + |e_i|$, $i = \overline{1,n}$ şi există măcar o linie a matricei pentru care ultima inegalitate se satisface cu semnul „>”.

Pentru implementare sunt suficiente patru tablouri $\bar{c}, \bar{d}, \bar{e}, \bar{b}$. Elementele α_i pot fi stocate în tabloul $\bar{e}$ peste mărimile e_i, iar elementele β_i şi componentele soluției x_i, în tabloul $\bar{b}$.

Pseudocodul algoritmului

Date de intrare:
n (de tip întreg) – ordinul sistemului liniar;
$d[n]$ ($c[n]$; $e[n]$) - tablou (cu elemente de tip real; indici de la 1 la n (de la 2 la n; de la 1 la $n-1$)) ce conține diagonala principală (inferioară; superioară) a matricei coeficienților sistemului;
$b[n]$ - tablou (cu elemente de tip real; indici de la 1 la n) ce conține componentele vectorului termenilor liberi ai sistemului.
Date de ieşire:
$b[n]$ - tablou (cu elemente de tip real; indici de la 1 la n) ce conține componentele soluției sistemului.

```
întreg k
real h
  dacă d_1 = 0 atunci {se afişează mesajul "Nu este posibilă aplicarea algoritmului"; STOP }
  e_1 := -e_1/d_1 ; b_1 := b_1/d_1
  pentru k := 2,n-1 execută
    {
      h := d_k + c_k e_{k-1}
      dacă h = 0 atunci {se afişează mesajul "Nu este posibilă aplicarea algoritmului"; STOP }
      e_k := -e_k/h; b_k := (b_k - c_k b_{k-1})/h
    }
  h := d_n + c_n e_{n-1}
  dacă h = 0 atunci {se afişează mesajul "Nu este posibilă aplicarea algoritmului"; STOP }
  b_n := (b_n - c_n b_{n-1})/h
  pentru k := n-1,1 execută b_k := e_k b_{k+1} + b_k
pentru k := 1,n scrie b_k
```

Remarcă. Poate fi utilizat şi un algoritm analog cu cel expus mai sus, cunoscut sub numele *metoda parcurgerii de stânga*:

$$\alpha_n = -c_n/d_n,\ \beta_n = b_n/d_n\ \ (d_n \neq 0),\ h_i = d_i + e_i\alpha_{i+1},\ \alpha_i = -c_i/h_i,\ \beta_i = (b_i - e_i\beta_{i+1})/h_i,\ i = \overline{n-1,2},$$
$$h_1 = d_1 + e_1\alpha_2,\ x_1 = (b_1 - e_1\beta_2)/h_1\ \ (h_i \neq 0,\ i = \overline{1,n-1}),\ x_i = \alpha_i x_{i-1} + \beta_i,\ i = \overline{2,n}.$$

B) *Calculul determinantului şi a inversei matricei* A

Eliminarea gaussiană poate fi utilizată şi pentru calculul numeric al valorii determinantului $\det A$ şi a inversei A^{-1} a matricei coeficienților A [1-5; 10-11; 15-16; 21-26].

Dacă la căutarea pivoților nu se recurge la permutări de linii şi coloane, determinantul matricei inițiale A este egal cu determinantul matricei $A^{(n)}$, obținută după cele $n-1$ etape al procesului de triunghiularizare, şi întrucât $A^{(n)}$ este triunghiulară, $\det A$ este produsul elementelor diagonale ale acesteia. Pentru o mai bună exactitate a calculelor se recomandă utilizarea tehnicilor de pivotare, iar în acest caz, $\det A$ se va calcula prin formula $\det A = (-1)^p \prod_{k=1}^{n} a_{kk}^{(k)}$, unde p reprezintă numărul total de permutări de linii şi coloane [3; 16].

Problema determinării matricei inverse A^{-1} este echivalentă cu rezolvarea a n sisteme de n ecuații

liniare cu n necunoscute, având ca termeni liberi coloanele matricei unitate I, iar ca vectori necunoscuți coloanele matricei inverse: $Ax^{(i)} = e_i,\ i = \overline{1,n}$ [3; 16]. Aceste sisteme au aceeaşi matrice A şi pot fi rezolvate simultan cu metoda eliminării lui Gauss. Totuşi, pentru soluționarea problemei propuse este mai rațional să se aplice transformările de la eliminarea lui Gauss-Jordan. Vom considera matricea de dimensiune $n \times 2n$, formată din alăturarea matricei unitate I la matricea A, urmând a aplica transformările matricei A astfel, încât să o aducem la matricea unitate I:

$$[A \mid I] \to [I \mid B].$$

Dacă procedura de eliminare Gauss-Jordan nu necesită permutări de coloane, atunci $B = A^{-1}$. În cazul general (necesar din punctul de vedere al stabilității numerice) matricea inversă se obține din matricea B printr-o permutare de linii, corespunzătoare permutărilor de coloane efectuate la căutarea pivoților în operațiile de eliminare. Permutările de linii în matricea B se efectuează în ordine inversă față de permutările de coloane efectuate în matricea A.

Implementarea algoritmului se face simplu, ținând cont de cele menționate mai sus.

În **metoda factorizării** LU matricea A a sistemului (5.1) se reprezintă sub forma unui produs de două matrici [2-4; 8; 10-11; 13; 15; 20-28]:

$$A = LU, \tag{5.6}$$

unde $L = \{l_{ij}\}_{i,j=1}^{n}$ $(i \geq j)$ este o matrice inferior triunghiulară, iar $U = \{u_{ij}\}_{i,j=1}^{n}$ $(i \leq j)$ o matrice superior triunghiulară.

În acest mod, rezolvarea sistemului (5.1) se reduce la rezolvarea, mai întâi, a sistemului inferior triunghiular $L\bar{y} = \bar{b}$, de unde se obține $\bar{y}$, urmată de rezolvarea sistemului superior triunghiular $U\bar{x} = \bar{y}$, de unde se obține $\bar{x}$, soluția sistemului dat. Relația (5.6) reprezintă un sistem de n^2 ecuații cu $n^2 + n$ necunoscute, deci, trebuie precizate n dintre necunoscute. Dacă se impune condiția ca elementele de pe diagonala principală a matricei L să fie egale cu 1 ($l_{ii} = 1,\ i = \overline{1,n}$), metoda de descompunere (5.6) se numeşte factorizarea Doolittle. Alte alegeri frecvent utilizate sunt date de factorizarea Crout [2, p.147] când se aleg $u_{ii} = 1,\ i = \overline{1,n}$ şi factorizarea Cholesky când $U = L^T$, deci $l_{ii} = u_{ii},\ i = \overline{1,n}$. Determinarea matricelor L şi U se face simultan.

Metoda factorizării Doolittle ($l_{ii} = 1,\ i = \overline{1,n}$)

Relațiile pentru calculul elementelor nenule ale matricelor L şi U:

$$u_{ik} = a_{ik} - \sum_{m=1}^{i-1} l_{im} u_{mk},\ i = \overline{1,k},\quad l_{ik} = \left(a_{ik} - \sum_{m=1}^{k-1} l_{im} u_{mk} \right) \Big/ u_{kk},\ i = \overline{k+1,n},\quad k = \overline{1,n}.$$

Relațiile pentru calculul soluțiilor sistemelor cu matrice triunghiulare:

$$y_1 = b_1,\ y_k = b_k - \sum_{m=1}^{k-1} l_{km} y_m,\ k = \overline{2,n},\quad x_n = y_n / u_{nn},\ x_k = \left(y_k - \sum_{m=k+1}^{n} u_{km} x_m \right) \Big/ u_{kk},\ k = \overline{n-1,1}.$$

Metoda factorizării Crout ($u_{ii} = 1,\ i = \overline{1,n}$)

Relațiile pentru calculul elementelor nenule ale matricelor L şi U:

$$u_{ik}=\left(a_{ik}-\sum_{m=1}^{i-1}l_{im}u_{mk}\right)\Big/l_{ii},\ i=\overline{1,k-1},\ \ l_{ik}=a_{ik}-\sum_{m=1}^{k-1}l_{im}u_{mk},\ i=\overline{k,n},\ \ k=\overline{1,n}.$$

Relațiile pentru calculul soluțiilor sistemelor cu matrice triunghiulare:

$$y_1=b_1/l_{11},\ y_k=\left(b_k-\sum_{m=1}^{k-1}l_{km}y_m\right)\Big/l_{kk},\ k=\overline{2,n},\ \ x_n=y_n,\ x_k=y_k-\sum_{m=k+1}^{n}u_{km}x_m,\ k=\overline{n-1,1}.$$

Metoda factorizării Cholesky ($l_{ii}=u_{ii},\ i=\overline{1,n}$) [2-5; 10-13; 16; 20-24; 26]

Relațiile pentru calculul elementelor nenule ale matricelor L şi U:

$$l_{kk}=\sqrt{a_{kk}-\sum_{m=1}^{k-1}l_{km}^2},\ l_{ik}=\left(a_{ik}-\sum_{m=1}^{k-1}l_{im}l_{km}\right)\Big/l_{kk},\ i=\overline{k+1,n},\ \ k=\overline{1,n}.$$

Relațiile pentru calculul soluțiilor sistemelor cu matrice triunghiulare:

$$y_1=b_1/l_{11},\ y_k=\left(b_k-\sum_{m=1}^{k-1}l_{km}y_m\right)\Big/l_{kk},\ k=\overline{2,n},\ \ x_n=y_n/l_{nn},\ x_k=\left(y_k-\sum_{m=k+1}^{n}l_{mk}x_m\right)\Big/l_{kk},\ k=\overline{n-1,1}.$$

Elementele nenule ale matricelor L şi U (cu excepția celor egale cu 1) vor fi stocate în zonele rezervate elementelor corespunzătoare ale matricei A: $a_{ij}:=l_{ij},i\ge j,\ a_{ij}:=u_{ij},i<j,\ i,j=\overline{1,n}$. Soluțiile $\overline{y}$ şi $\overline{x}$ ale sistemelor cu matrice triunghiulare se vor memora în vectorul $\overline{b}$.

Pseudocodul algoritmilor

Date de intrare:
n (de tip întreg) – ordinul matricei A;
$a[n,n]$ – tablou bidimensional (cu elemente de tip real; indici de la 1 la n) ce conține componentele matricei A;
$b[n]$ – tablou (cu elemente de tip real; indici de la 1 la n) ce conține componentele vectorului termenilor liberi ai sistemului.
Date de ieşire:
$b[n]$ – tablou (cu elemente de tip real; indici de la 1 la n) ce conține componentele soluției sistemului;
întreg i,k,m

Rezolvarea SEAL cu factorizarea Doolittle

pentru $k:=\overline{1,n}$ execută
{
  pentru $i:=\overline{1,k}$ execută
  {
    pentru $m:=\overline{1,i-1}$ execută $a_{ik}:=a_{ik}-a_{im}a_{mk}$
  }
  pentru $i:=\overline{k+1,n}$ execută
  {
    pentru $m:=\overline{1,k-1}$ execută $a_{ik}:=a_{ik}-a_{im}a_{mk}$
    $a_{ik}:=a_{ik}/a_{kk}$
  }
}
pentru $k:=\overline{2,n}$ execută
  {
    pentru $m:=\overline{1,k-1}$ execută $b_k:=b_k-a_{km}b_m$
  }
$b_n:=b_n/a_{nn}$
pentru $k:=\overline{n-1,1}$ execută
  {

Rezolvarea SEAL cu factorizarea Crout

pentru $k:=\overline{1,n}$ execută
 {
  pentru $i:=\overline{1,k-1}$ execută
  {
    pentru $m:=\overline{1,i-1}$ execută $a_{ik}:=a_{ik}-a_{im}a_{mk}$
    $a_{ik}:=a_{ik}/a_{ii}$
  }
  pentru $i:=\overline{k,n}$ execută
  {
    pentru $m:=\overline{1,k-1}$ execută $a_{ik}:=a_{ik}-a_{im}a_{mk}$
  }
 }
$b_1:=b_1/a_{11}$
pentru $k:=\overline{2,n}$ execută
 {
    pentru $m:=\overline{1,k-1}$ execută $b_k:=b_k-a_{km}b_m$
    $b_k:=b_k/a_{kk}$
 }
pentru $k:=\overline{n-1,1}$ execută

pentru $m := \overline{k+1,n}$ execută $b_k := b_k - a_{km} b_m$
$b_k := b_k / a_{kk}$
}
pentru $k := \overline{1,n}$ scrie b_k
	{
	pentru $m := \overline{k+1,n}$ execută $b_k := b_k - a_{km} b_m$
	}
	pentru $k := \overline{1,n}$ scrie b_k

Rezolvarea SEAL cu factorizarea Cholesky

pentru $k := \overline{1,n}$ execută
{
pentru $m := \overline{1,k-1}$ execută $a_{kk} := a_{kk} - a_{km}^2$
$a_{kk} := \sqrt{a_{kk}}$
pentru $i := \overline{k+1,n}$ execută
{
pentru $m := \overline{1,k-1}$ execută $a_{ik} := a_{ik} - a_{im} a_{km}$
$a_{ik} = a_{ik} / a_{kk}$
}
}
$b_1 := b_1 / a_{11}$
pentru $k := \overline{2,n}$ execută
{
pentru $m := \overline{1,k-1}$ execută $b_k := b_k - a_{km} b_m$
$b_k := b_k / a_{kk}$
}
$b_n := b_n / a_{nn}$
pentru $k := \overline{n-1,1}$ execută
{
pentru $m := \overline{k+1,n}$ execută $b_k := b_k - a_{mk} b_m$
$b_k := b_k / a_{kk}$
}
pentru $k := \overline{1,n}$ scrie b_k

Matricele L şi U, fiind triunghiulare, este indicat ca ele să fie localizate într-un tablou unidimensional, economisindu-se astfel memoria ce s-ar rezerva pentru elementele nule ale respectivelor matrice.

Pentru a evita necesitatea ca toți minorii principali să fie nenuli, factorizarea LU poate fi implementată eficient folosind tehnica pivotului parțial, caz în care $PA = LU$, iar soluția se obține rezolvând sistemul echivalent $LU\bar{x} = P\bar{b}$ (de fapt, se rezolvă sistemele $L\bar{y} = P\bar{b}$ şi $U\bar{x} = \bar{y}$), unde P este matricea care conține toate permutările de linii efectuate asupra matricei A pe parcursul procesului de factorizare.

Analiza algoritmilor

A.e.G. (în varianta fără pivotare) poate genera instabilități numerice chiar dacă nu se întâlneşte nici un element pivot nul, din cauza erorilor de rotunjire. Deşi stabilitatea numerică a a.e.G cu pivotare parțială este mai bună, totuşi, erorile de rotunjire pot afecta grav soluția în cazul sistemelor slab condiționate.

În tabelul de mai jos se face o analiză comparativă a complexității de calcul a algoritmilor direcți descrişi mai sus (vom utiliza mărimile $N_1 = 2n^3/3 + O(2n^2)$, $N_2 = 0.5n^2 - 0.5n$; $N_3 = n^3/3 - n^2/2 + n/6$; $N_4 = n^3 + O(n^2)$):

	a.e.G			**a.e.GJ**			**descompunerea** LU		**Alg. metodei parcurgerii** (matrice tridiagonală)
	fără pivotare	pivotare parțială	pivotare totală	fără pivotare	pivotare parțială	pivotare totală	Crout sau Doolittle	Cholesky	
Nr. operaț. aritmetice	N_1	N_1+N_2	N_1+N_3	N_4	N_4+N_2	N_4+N_3	N_1	$n^3/3+O(n^2)$	$O(8n)$
Volumul de memorie	$O(n^2)$			$O(n^2)$			$O(n^2)$	$O(0.5n^2)$	$O(3n)$

Pentru a obține inversa matricei, conform algoritmului prezentat mai sus, este necesar un efort de calcul de $O(2n^3)$ operații aritmetice şi un volum de memorie de $O(2n^2)$. Vom menționa că inversarea matricei se reduce la rezolvarea a n SEAL, iar rezolvarea acestora necesită doar de trei ori mai multe operații decât rezolvarea unui singur sistem.

Dacă o matrice nesingulară A admite o descompunere (factorizare) LU, atunci elementele de pe diagonalele principale ale matricelor L şi U sunt nenule. În cazul în care toți minorii principali ai matricei A sunt nenuli, iar elementele de pe diagonala principală a matricei L sau U sunt fixate (şi nenule), există o unică factorizare LU a matricei A [2, p.145].

Un avantaj al metodei factorizării LU este posibilitatea rezolvării sistemului cu mai multe coloane de termeni liberi cu o mare economie de efort de calcul. Dacă avem de rezolvat mai multe sisteme $A\bar{x}=\bar{b}^{(i)}, i=\overline{1,m}$, rezolvarea tuturora necesită $2mn^3/3+O(2mn^2)$ operații; dacă însă mai întâi se factorizează - rezolvarea unui sistem durează $2n^2+O(n)$, factorizarea durează $2n^3/3+O(n^2)$, iar per total avem $2n^3/3+O(2mn^2)$ operații.

Vom menționa că determinantul matricei inițiale A a sistemului (5.1) este egal cu produsul elementelor de pe diagonala principală a matricei A, obținută în urma procesului de construcție a factorizării LU.

Dacă matricea coeficienților A este simetrică ($a_{ij}=a_{ji}, i,j=\overline{1,n}$) şi pozitiv definită (pentru orice vector $\bar{x}\in\mathbb{R}^n$, $\bar{x}\neq 0$ avem $\bar{x}^T A\bar{x}>0$ sau echivalent, toți minorii principali sunt pozitivi), atunci ea admite o unică factorizare $A=LL^T$ (factorizarea Cholesky a matricei A), în care L este o matrice inferior triunghiulară cu toate elementele diagonale pozitive. Factorizarea Cholesky este o variantă a eliminării lui Gauss ce operează atât la stânga, cât şi la dreapta matricei, păstrând şi exploatând simetria. În condițiile impuse matricei A, algoritmul este stabil. Datorită factorizării Cholesky, matricele simetrice pozitiv definite pot fi descompuse în factori triunghiulari de două ori mai repede decât matricele generale. Mai mult, algoritmul de factorizare Cholesky defineşte un procedeu econom de a stabili dacă o matrice

simetrică este pozitiv definită: acesta îşi va întrerupe lucrul la un pas oarecare dacă va apărea situația în care este necesar să se extragă rădăcina pătrată dintr-un număr negativ sau ca urmare a împărțirii la zero.

Algoritmul de rezolvare a SEAL, bazat pe factorizarea lui Cholesky (cunoscut şi ca metoda rădăcinii pătrate), se poate utiliza pentru orice sistem liniar cu matrice nesingulară, prin transformarea sa într-un sistem echivalent cu matrice simetrică (se înmulțește relația (5.1) cu matricea transpusă A^T).

Vom remarca că calculul erorii reale ar necesita cunoaşterea soluției exacte a sistemului. De aceea, în practică se face doar o evaluare a erorii, folosind norma reziduului $\|\bar{r}\| = \|A\bar{x} - \bar{b}\| = \sqrt{\sum_{k=1}^{n} r_k^2}$, care însă poate avea o valoare mai mare sau mai mică decât eroarea reală.

♦ **Metode iterative**

Aceste metode permit determinarea unei aproximații a soluției sistemului (5.1) cu precizia dorită prin generarea unui şir de vectori care, în anumite condiții, converge către soluția exactă. Elementele acestui şir se calculează recursiv, iar procesul iterativ se opreşte după un număr finit de paşi, la îndeplinirea criteriului de stopare a calculelor [1-5; 8; 10-11; 13-16; 19-21; 23-28].

Erorile de rotunjire în metodele iterative au o influență cu mult mai redusă decât în cele directe. Drept urmare, chiar dacă soluția obținută prin metode iterative este afectată de erori de trunchiere, erori care nu apar în cazul metodelor directe, este totuşi posibil ca soluția obținută prin metodele iterative să fie mai precisă decât cea obținută prin metodele directe. În plus, dacă procesul iterativ este suficient de rapid convergent se poate obține o economie importantă de timp de calcul.

Pentru rezolvarea SEAL (5.1) considerăm clasa de metode iterative monopas [14; 23; 26]:

$$B_{k+1}\frac{\bar{x}^{(k+1)} - \bar{x}^{(k)}}{\tau_{k+1}} + A\bar{x}^{(k)} = \bar{b},\ k = 0,1,\ldots,$$

unde şirul $B_{k+1},\ k = 0,1,\ldots$ constă din matrice pătrate de ordinul n nesingulare, iar $\tau_{k+1} > 0$ sunt parametri reali care definesc metoda. Dacă matricele B_{k+1} şi parametrii τ_{k+1} nu variază pe parcursul iterațiilor (adică nu depind de k), atunci procesul iterativ se numeşte staționar. În caz contrar, considerăm că acesta este nestaționar.

Metode staționare

Metodele iterative staționare pot fi exprimate în forma simplificată $\bar{x}^{(k+1)} = M\bar{x}^{(k)} + \bar{c}$, unde k reprezintă pasul de iterare, iar M şi $\bar{c}$ nu depind de k, adică acestea rămân constante pe durata procesului iterativ [1-5; 8; 10-11; 13-16; 19-21; 23-28].

În continuare vom prezenta cele mai simple metode iterative staționare pentru rezolvarea sistemului (5.1) şi vom evidenția condițiile în care acestea converg. Să presupunem că elementele matricei coeficienților A a sistemului (5.1) satisfac condiției: $a_{ii} \neq 0$ $(i = \overline{1,n})$.

Metoda lui Jacobi (iterației simple, aproximațiilor succesive, deplasărilor simultane) utilizează următoarea relație de recurență $\bar{x}^{(k+1)} = M\bar{x}^{(k)} + \bar{c}$, unde $M = \{m_{ij}\}_{i,j=1}^{n}$, $\bar{c} = \{c_i\}_{i=1}^{n}$, $m_{ij} = 0$ pentru $i = j$, $m_{ij} = -a_{ij}/a_{ii}$ pentru $i \neq j$, iar $c_i = b_i/a_{ii}$. Procesul iterativ indicat se va scrie pe coordonate astfel:

$$x_i^{(k+1)} = \left(b_i - \sum_{j=1,\, j\neq i}^{n} a_{ij} x_j^{(k)} \right) \Big/ a_{ii},\ i = \overline{1,n},\ \ k = 0,1,\ldots\ (a_{ii} \neq 0),$$

în care $x_i^{(k+1)}$ reprezintă componenta i a noii aproximații, iar $x_j^{(k)}$ $(j = \overline{1,n},\ j \neq i)$ sunt valorile obținute la pasul precedent al procesului iterativ. Se observă că e necesar a păstra în memoria calculatorului toate componentele vectorului $\overline{x}^{(k)}$ atâta timp, cât se calculează aproximația $\overline{x}^{(k+1)}$.

Criteriul de stopare a procesului iterativ constă în evaluarea după fiecare iterație a erorii Cauchy $\left\| \overline{x}^{(k+1)} - \overline{x}^{(k)} \right\|$ şi întreruperea calculelor atunci când această valoare devine mai mică decât eroarea impusă ε [1-5; 8; 10-11; 13-16; 19-21; 23-28].

Metoda lui Gauss-Seidel (deplasărilor succesive) reprezintă o modificare a metodei lui Jacobi. Pentru calculul necunoscutelor la pasul curent, se utilizează, acolo unde este posibil, şi valorile necunoscutelor calculate chiar la acest pas, şi nu în exclusivitate toate valorile de la pasul anterior. Procesul iterativ se va scrie pe coordonate astfel:

$$x_i^{(k+1)} = \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right) \Big/ a_{ii},\ i = \overline{1,n},\ \ k = 0,1,\ldots\ (a_{ii} \neq 0).$$

Se observă că o componentă $x_i^{(k+1)}$ a soluției la pasul $k+1$ este calculată în funcție de componentele precedente $x_1^{(k+1)},\ldots,x_{i-1}^{(k+1)}$, deja calculate la pasul curent şi de următoarele componente $x_{i+1}^{(k)},\ldots,x_n^{(k)}$, calculate la pasul precedent. Algoritmul nu necesită păstrarea "vechii" componente $x_i^{(k)}$, după ce cea "nouă" a fost calculată, de aceea $x_i^{(k+1)}$ se poate plasa în memorie în aceeaşi locație ca şi vechea valoare $x_i^{(k)}$. Astfel, algoritmul lui Gauss-Seidel nu necesită spațiu pentru memorarea decât a unui vector soluție, spre deosebire de algoritmul lui Jacobi, unde trebuiau memorați atât $x_i^{(k+1)}$, cât şi $x_i^{(k)}$.

Pseudocodul algoritmilor

Date de intrare:
- n (de tip întreg) – ordinul sistemului liniar;
- $a[n,n]$ - tablou bidimensional (cu elemente de tip real; indici de la 1 la n) ce conține componentele matricei A;
- $b[n]$ - tablou (cu elemente de tip real; indici de la 1 la n) ce conține componentele vectorului termenilor liberi ai sistemului;
- ε (de tip real) – eroarea admisă;
- q (de tip real) – constanta lui Lipschitz.

Date de ieşire:
- $x[n]$ - tablou (cu elemente de tip real; indici de la 1 la n) ce conține componentele soluției sistemului;
- k (de tip real) – numărul de iterații efectuate pentru a atinge exactitatea dorită.

Metoda lui Jacobi

```
întreg i, j, k
real ε1, s, er
tablou real z[n]
    k := 0; ε1 := (1 − q)ε/q
    pentru i := 1,n execută  x_i := b_i / a_ii
    execută
      {
        er := 0
        pentru i := 1,n execută
```

Metoda lui Gauss-Seidel

```
întreg i, j, k
real ε1, s, er
    k := 0; ε1 := (1 − q)ε/q
    pentru i := 1,n execută  x_i := b_i / a_ii
    execută
      {
        er := 0
        pentru i := 1,n execută
          {
```

```
        {
          s := b_i
          pentru j := 1,n execută  s := s − a_ij x_j
          s := s + a_ii x_i ; z_i := s/a_ii ; s := |z_i − x_i|
          dacă er < s atunci er := s
        }
      pentru i := 1,n execută  x_i := z_i
      k := k + 1
    }
  cât timp er ≥ ε_1
pentru i := 1,n scrie  x_i
scrie k
```

```
          s := b_i
          pentru j := 1,n execută  s := s − a_ij x_j
          s := (s + a_ii x_i)/a_ii ; er := er + (s − x_i)^2 ; x_i := s
        }
      k := k + 1
      er := √er
    }
  cât timp er ≥ ε_1
pentru i := 1,n scrie  x_i
scrie k
```

Analiza algoritmilor

Este suficient să aibă loc oricare dintre condițiile următoare pentru ca să se asigure convergența metodei lui Jacobi sau Gauss-Seidel (indiferent de alegerea aproximației inițiale $\bar{x}^{(0)}$) [2, p.185; 14; 23]:

a) Matricea A a sistemului (5.1) este strict diagonal dominantă pe linii, adică $|a_{ii}| > \sum_{j=1, j\neq i}^{n} |a_{ij}|,\ i = \overline{1,n}$;

b) Matricea A a sistemului (5.1) este strict diagonal dominantă pe coloane, adică $|a_{jj}| > \sum_{i=1, i\neq j}^{n} |a_{ij}|$, $j = \overline{1,n}$;

c) Matricea A a sistemului (5.1) satisface relația $\sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} (a_{ij}/a_{ii})^2 < 1$;

d) Matricea A a sistemului (5.1) este simetrică şi pozitiv definită (Gauss-Seidel).

Pentru estimarea erorii care se comite la înlocuirea soluției exacte $\bar{x}^*$ a sistemului (5.1) cu o aproximație $\bar{x}^{(k+1)}$ se utilizează inegalitatea [23, p.94]

$$\left\|\bar{x}^{(k+1)} - \bar{x}^*\right\| \le \frac{q}{1-q}\left\|\bar{x}^{(k+1)} - \bar{x}^{(k)}\right\|,\ k = 0,1,\ldots\ (q := \|M\|),$$

în cazul în care aproximația se obține conform metodei lui Jacobi, şi, respectiv, inegalitatea [23, p.107]

$$\left\|\bar{x}^{(k+1)} - \bar{x}^*\right\| \le \frac{\mu}{1-q}\left\|\bar{x}^{(k+1)} - \bar{x}^{(k)}\right\|,\ k = 0,1,\ldots\ (\mu := \|M_U\|,\ q := \|M\|),$$

atunci când se aplică metoda lui Gauss-Seidel. Matricele M şi M_U sunt definite în modul următor:

$$M = \{m_{ij}\}_{i,j=1}^{n},\ M_U = \{m_{ij}^U\}_{i,j=1}^{n},\ m_{ij} = \begin{cases} 0 \text{ pentru } i = j \\ -a_{ij}/a_{ii} \text{ pentru } i \neq j \end{cases}, \text{ iar } m_{ij}^U = \begin{cases} 0 \text{ pentru } i \ge j \\ -a_{ij}/a_{ii} \text{ pentru } i < j \end{cases}.$$

În estimațiile de mai sus se utilizează următoarele norme operatoriale de matrice compatibile cu norme vectoriale de pe $\mathbb{R}^n$:

$$\|A\|_1 := \max_{1\le j\le n} \sum_{i=1}^{n} |a_{ij}| \qquad \|\bar{x}\|_1 := \sum_{i=1}^{n} |x_i|,$$

$$\|A\|_\infty := \max_{1\le i\le n} \sum_{j=1}^{n} |a_{ij}| \qquad \|\bar{x}\|_\infty := \max_{1\le i\le n} |x_i|.$$

Constanta q se va alege astfel încât $q < 1$.

Metodele iterative prezentate necesită $O(n^2+2n)$ operații aritmetice la fiecare iterație. Pentru memorarea matricei sistemului, a vectorului termenilor liberi şi a soluției sunt necesare n^2+2n locații de memorie. În plus, la algoritmul lui Jacobi mai sunt necesare n locații pentru memorarea soluției obținute la pasul curent. Prin urmare, metodele iterative devin avantajoase dacă numărul k de iterații nu este prea mare. Spre exemplu, o comparație cu algoritmul de eliminare a lui Gauss conduce la concluzia că pentru a avea un număr apropiat de operații, este necesar ca numărul de iterații k să satisfacă inegalitatea $k \le 2n/3$. Numărul de iterații este influențat de calitatea aproximației inițiale $\bar{x}^{(0)}$, de modul în care este condiționată matricea A a coeficienților sistemului şi de norma matricei de iterație M.

Avantajul esențial al metodei lui Gauss-Seidel îl constituie creşterea vitezei de convergență, datorită reducerii numărului total de iterații necesare. În cazul matricelor simetrice şi pozitiv definite, metoda lui Gauss-Seidel este de aproximativ două ori mai rapidă decât metoda lui Jacobi. Acest avantaj, corelat şi cu necesitatea memorării unui singur vector soluție, face ca metoda lui Gauss-Seidel să fie preferabilă metodei lui Jacobi.

O metodă de stocare compactă a matricei simetrice

Matricea simetrică şi pozitiv definită poate fi memorată folosind următoarele tablouri unidimensionale [19, p.7]:

- $u[n]$ - conține elementele diagonalei principale ale matricei A. Întrucât matricea este pozitiv definită, elementele $u_i, i=\overline{1,n}$ sunt strict pozitive;
- $c[]$ - conține elementele nenule ale matricei A care sunt amplasate sub diagonala principală. Elementele $c_i, i=1,2,...$ sunt ordonate în sensul în care urmează pe liniile lui A;
- $t[]$ - are acelaşi număr de elemente ca şi $c[]$, iar elementul t_i al acestuia conține indicele coloanei matricei A în care se află elementul c_i al tabloului $c[]$;
- $s[n+1]$ - elementul $s_i, i=\overline{1,n}$ indică din care poziție a tabloului $c[]$ începe linia cu numărul i al matricei A. Valoarea ultimului element s_{n+1} reprezintă numărul de elemente ale tabloului $c[]$ incrementat cu o unitate. Diferența $s_{i+1}-s_i$ $(i=\overline{1,n})$ este egală cu numărul de elemente nenule ale liniei i a matricei A, amplasate până la elementul de pe diagonala principală.

Remarcă. Tabloul $s[n+1]$ nu va conține informații referitoare la prima linie a matricei A, precum şi la acele linii i pentru care toate elementele a_{ij} $(i>j)$ sunt nule. În acest caz, respectivul element s_i va obține valoarea lui s_k $(k>i)$, unde k este prima linie a matricei A ce urmează după linia cu numărul i şi care conține elemente nenule.

Vom descrie în continuare un algoritm eficient (mai ales în cazul în care matricea A este rară) care realizează calculul produsului matricei simetrice A cu un vector $\bar{x}$ [19, p.8]:

Date de intrare: $n, \overline{x}, \overline{u}, \overline{c}, \overline{s}, \overline{t}$
Date de ieşire: $\overline{z} = A\overline{x}$
pentru $i := \overline{1,n}$ execută { $z_i := u_i x_i$ }
pentru $i := \overline{1,n}$ execută
{
pentru $j := \overline{s_i, s_{i+1} - 1}$ execută { $z_i := z_i + c_j x_{t_j}$; $z_{t_j} := z_{t_j} + c_j x_i$ }
}

Metode nestaționare.

În cele ce urmează, pentru orice doi vectori coloană $\overline{u} = (u_1, ..., u_n)^T$ şi $\overline{v} = (v_1, ..., v_n)^T$ vom nota produsul lor scalar cu $\langle \overline{u}, \overline{v} \rangle := \overline{v}^T \overline{u} = \sum_{i=1}^{n} u_i v_i$.

Vom considera că matricea coeficienților A este simetrică şi pozitiv definită. Problema de aflare a soluției sistemului de ecuații (5.1) este echivalentă cu problema (variațională) determinării punctului de minim al funcționalei [4, p.48; 14; 21; 23]

$$F(\overline{x}) = \frac{1}{2} \overline{x}^T A \overline{x} - \overline{x}^T \overline{b} \tag{5.7}$$

($A, \overline{b}$ - componentele sistemului (5.1)). Astfel, soluția sistemului (5.1) coincide cu punctul de minim al funcționalei strict convexe (5.7).

Vom formula principiul de bază al metodelor nestaționare care vor fi prezentate în continuare. Fie $\overline{x}^{(k)}$ o aproximație a soluției problemei de minimizare şi presupunem că vectorul rezidual $\overline{r}^{(k)} := \overline{b} - A\overline{x}^{(k)} \neq \overline{0}$. Alegem o direcție de căutare $\overline{d}^{(k)} \neq \overline{0}$ şi determinăm noua aproximație $\overline{x}^{(k+1)} = \overline{x}^{(k)} + \alpha \overline{d}^{(k)}$ ($\alpha \in \mathbb{R}$) a punctului de minim $\overline{x}^*$ al funcționalei $F(\overline{x})$, astfel încât $F(\overline{x}^{(k+1)}) = \min_{\alpha \in \mathbb{R}} F(\overline{x}^{(k)} + \alpha \overline{d}^{(k)})$. Soluția ultimei probleme de extrem se obține pentru $\alpha := \alpha_k = \left((\overline{d}^{(k)})^T \overline{r}^{(k)} \right) \Big/ \left((\overline{d}^{(k)})^T A \overline{d}^{(k)} \right)$. Valoarea funcționalei F descreşte de-a lungul dreptei $\overline{x}^{(k)} + \alpha \overline{d}^{(k)}$ dacă şi numai dacă vectorul rezidual $\overline{r}^{(k)}$ nu este ortogonal pe direcția de deplasare $\overline{d}^{(k)}$. Dacă $\overline{r}^{(k+1)} := \overline{b} - A\overline{x}^{(k+1)} \neq \overline{0}$, atunci se repetă procesul de căutare liniară utilizând o nouă direcție de căutare $\overline{d}^{(k+1)}$.

Diverşii algoritmi de minimizare a funcționalei (5.7) se obțin prin precizarea metodei de selecție a direcției $\overline{d}^{(k)}$ la pasul k .

Metoda pasului descendent (coborârii rapide)

Fiind dată aproximația $\overline{x}^{(k)}$, se ştie că cea mai rapidă descreştere a valorii funcționalei (5.7) este în direcția cu semn schimbat a gradientului ∇F al acesteia în $\overline{x}^{(k)}$. Deci, alegerea optimă locală a direcției de deplasare este $\overline{d}^{(k)} = -\nabla F(\overline{x}^{(k)})$. Deoarece $\nabla F(\overline{x}) = A\overline{x} - \overline{b}$, avem $\overline{d}^{(k)} = \overline{r}^{(k)}$. Aşadar, avem următorul algoritm de pas descendent [4; 11; 19-20].

La pasul de inițializare se alege aproximația inițială $\overline{x}^{(0)}$ şi se determină vectorul rezidual inițial $\overline{r}^{(0)} = \overline{b} - A\overline{x}^{(0)}$. Formula de calcul a aproximațiilor este:

$$\overline{x}^{(k+1)} = \overline{x}^{(k)} + \alpha_k \overline{r}^{(k)}, \; k = 0, 1, ...,$$

unde pasul α_k se calculează în baza relațiilor $\alpha_k = \left((\bar{r}^{(k)})^T \bar{r}^{(k)}\right) \Big/ \left((\bar{r}^{(k)})^T \bar{q}^{(k)}\right)$, $\bar{q}^{(k)} = A\bar{r}^{(k)}$. Actualizarea vectorului rezidual $\bar{r}^{(k+1)}$ este rațional să se facă în baza formulei recursive $\bar{r}^{(k+1)} = \bar{r}^{(k)} - \alpha_k \bar{q}^{(k)}$. La fiecare iterație aceasta salvează o înmulțire matrice-vector $A\bar{x}^{(k+1)}$, deoarece produsul $A\bar{r}^{(k)}$ este deja calculat. În calitate de criteriu de stopare a calculelor poate fi utilizată una dintre condițiile:

$$\text{a) } \left\|\bar{r}^{(k+1)}\right\| < \varepsilon \quad \text{sau} \quad \text{b) } \left\|\bar{x}^{(k+1)} - \bar{x}^{(k)}\right\| < \varepsilon,$$

unde $\varepsilon > 0$ este o toleranță suficient de mică. La îndeplinirea unei condiții dintre cele două menționate, vectorul $\bar{x}^{(k+1)}$ se va considera soluție aproximativă a sistemului (5.1), calculată cu exactitatea ε.

Pseudocodul algoritmului

Date de intrare:
- n (de tip întreg) – ordinul sistemului liniar;
- ε (de tip real) – exactitatea de calcul;
- $c[], u[n]$ - tablouri (cu elemente de tip real) ce conțin elementele a_{ij} $(i \geq j)$ ale matricei simetrice A a coeficienților sistemului (a se vedea secțiunea *O metodă de stocare compactă a matricei simetrice*);
- $t[], s[n+1]$ - tablouri (cu elemente de tip întreg) cu ajutorul cărora se face adresarea elementelor matricei simetrice A (a se vedea secțiunea *O metodă de stocare compactă a matricei simetrice*);
- $b[n]$ - tablou (cu elemente de tip real; indici de la 1 la n) ce conține componentele vectorului termenilor liberi ai sistemului;
- $x[n]$ - tablou (cu elemente de tip real; indici de la 1 la n) ce conține componentele aproximației inițiale ale sistemului.

Date de ieşire:
- $x[n]$ - tablou (cu elemente de tip real; indici de la 1 la n) ce conține componentele soluției sistemului ;
- $r[n]$ - tablou (cu elemente de tip real; indici de la 1 la n) ce conține componentele vectorului reziduu;
- k - numărul de iterații efectuate;

întreg i, j, k
real $\rho 1, nr, \alpha$
tablou real $r[n], q[n]$

pentru $i := \overline{1,n}$ execută $\{ r_i := b_i - u_i x_i \}$
pentru $i := \overline{1,n}$ execută
{
pentru $j := \overline{s_i, s_{i+1} - 1}$ execută $\{ r_i := r_i - c_j x_{t_j};\ r_{t_j} := r_{t_j} - c_j x_i \}$
}
$\rho 1 := 0$
pentru $i := \overline{1,n}$ execută $\rho 1 := \rho 1 + r_i^2$
$k := 0;\ nr := \sqrt{\rho 1}$
cât timp $nr \geq \varepsilon$ execută
{
aici urmează secvența (■) de la pagina 59 (vectorul $\bar{z}$ se substituie cu vectorul $\bar{q}$, iar $\bar{x}$ cu $\bar{r}$)
$\alpha := 0$
pentru $i := \overline{1,n}$ execută $\alpha := \alpha + r_i q_i$
$\alpha := \rho 1 / \alpha;\ \rho 1 := 0;\ nr := 0$
pentru $i := \overline{1,n}$ execută $\{ x_i := x_i + \alpha r_i;\ r_i := r_i - \alpha q_i;\ \rho 1 := \rho 1 + r_i^2 \}$
$k := k + 1;\ nr := \sqrt{\rho 1}$
}
pentru $i := \overline{1,n}$ scrie x_i
pentru $i := \overline{1,n}$ scrie r_i
scrie k

Analiza algoritmului

În cazul în care matricea coeficienților A este simetrică şi pozitiv definită, metoda converge pentru orice aproximație inițială $\bar{x}^{(0)}$. La fiecare iterație a algoritmului se calculează un produs matrice vector şi două produse scalare. Viteza de convergență este a unei progresii geometrice [21, p.446]. Pentru probleme slab condiționate viteza de convergență a algoritmului scade esențial.

Metoda gradienților conjugați

Pentru a evita comportarea în zigzag ce apare la aplicarea algoritmului pasului descendent trebuie să considerăm o altă metodă de determinare a direcției de deplasare. Conform metodei gradienților conjugați [4, p.61; 20, p.294; 23, p.119] selecția direcțiilor $\bar{d}^{(k)}$ se face astfel, încât $(\bar{d}^{(k)})^T A\bar{d}^{(j)} = \bar{0}$ pentru toți $j = \overline{0,k-1}$ şi $(\bar{d}^{(k)})^T \bar{r}^k \neq \bar{0}$. Vectorii reziduali generați $\bar{r}^{(k)}$ sunt ortogonali doi câte doi $(\bar{r}^{(k)})^T \bar{r}^{(j)} = \bar{0},\ j = \overline{0,k-1}$. Drept rezultat, obținem următoarea expresie pentru direcția $\bar{d}^{(k)} = \bar{r}^{(k)} + \beta_k \bar{d}^{(k-1)},\ \beta_k = \left((\bar{r}^{(k)})^T \bar{r}^{(k)}\right) \Big/ \left((\bar{r}^{(k-1)})^T \bar{r}^{(k-1)}\right)$, iar pentru lungimea pasului α_k avem $\alpha_k = \left((\bar{r}^{(k)})^T \bar{r}^{(k)}\right) \Big/ \left((\bar{d}^{(k)})^T A\bar{d}^{(k)}\right)$. Schema de calcul a metodei gradienților conjugați este următoarea [14]:

La pasul de inițializare se alege aproximația inițială $\bar{x}^{(0)}$, se determină vectorul rezidual inițial $\bar{r}^{(0)} = \bar{b} - A\bar{x}^{(0)}$ şi se pune $\bar{d}^{(0)} = \bar{r}^{(0)}$. Aproximațiile se calculează prin formula:

$$\bar{x}^{(k+1)} = \bar{x}^{(k)} + \alpha_k \bar{d}^{(k)},\ k = 0,1,\ldots,$$

unde lungimea pasului de deplasare este $\alpha_k = \rho_k \Big/ \left((\bar{d}^{(k)})^T \bar{q}^{(k)}\right),\ \rho_k = (\bar{r}^{(k)})^T \bar{r}^{(k)},\ \bar{q}^{(k)} = A\bar{d}^{(k)}$. Actualizarea vectorului rezidual $\bar{r}^{(k+1)}$ şi a direcției de deplasare $\bar{d}^{(k+1)}$ se face în baza formulelor recursive $\bar{r}^{(k+1)} = \bar{r}^{(k)} - \alpha_k \bar{q}^{(k)},\ \bar{d}^{(k+1)} = \bar{r}^{(k+1)} + \beta_k \bar{d}^{(k)},\ \beta_k = \rho_{k+1}/\rho_k$. În calitate de criteriu de stopare a calculelor poate fi utilizată una dintre condițiile:

$$\text{a) } \left\|\bar{r}^{(k+1)}\right\| < \varepsilon \quad \text{sau} \quad \text{b) } \left\|\bar{x}^{(k+1)} - \bar{x}^{(k)}\right\| < \varepsilon,$$

unde $\varepsilon > 0$ este o toleranță suficient de mică. La îndeplinirea uneia dintre condițiile menționate, vectorul $\bar{x}^{(k+1)}$ se va considera soluție aproximativă a sistemului (5.1), calculată cu exactitatea ε.

Pseudocodul algoritmului

Date de intrare:
- n (de tip întreg) – ordinul sistemului liniar;
- ε (de tip real) – exactitatea de calcul;
- $c[], u[n]$ – tablouri (cu elemente de tip real) ce conțin elementele a_{ij} $(i \geq j)$ ale matricei simetrice A a coeficienților sistemului (a se vedea secțiunea *O metodă de stocare compactă a matricei simetrice*);
- $t[], s[n+1]$ – tablouri (cu elemente de tip întreg) cu ajutorul cărora se face adresarea elementelor matricei simetrice A (a se vedea secțiunea *O metodă de stocare compactă a matricei simetrice*);
- $b[n]$ – tablou (cu elemente de tip real; indici de la 1 la n) ce conține componentele vectorului termenilor liberi ai sistemului;
- $x[n]$ – tablou (cu elemente de tip real; indici de la 1 la n) ce conține componentele aproximației inițiale a sistemului.

Date de ieşire:
- $x[n]$ – tablou (cu elemente de tip real; indici de la 1 la n) ce conține componentele soluției sistemului ;
- $r[n]$ – tablou (cu elemente de tip real; indici de la 1 la n) ce conține componentele vectorului reziduu;
- k – numărul de iterații efectuate;

```
întreg i, j, k
real ρ0, ρ1, nr, α, β
tablou real r[n], q[n], d[n]
  aici urmează secvența (▲) de la pagina 60
  cât timp nr ≥ ε execută
    {
      dacă k = 0 atunci { pentru i := 1,n execută d_i := r_i }
      altfel
        {
          β := ρ1/ρ0
          pentru i := 1,n execută d_i := β d_i − r_i
        }
      aici urmează secvența (■) de la pagina 59 (vectorul z se substituie cu vectorul q, iar x cu d)
      β := 0
      pentru i := 1,n execută β := β + d_i q_i
      α := ρ1/β; ρ0 := ρ1; ρ1 := 0; nr := 0
      pentru i := 1,n execută { x_i := x_i + α d_i; r_i := r_i − α q_i; ρ1 := ρ1 + r_i^2 }
      k := k + 1; nr := √ρ1
    }
pentru i := 1,n scrie x_i
pentru i := 1,n scrie r_i
scrie k
```

Analiza algoritmului

Întrucât vectorii reziduali $\bar{r}^{(k)}$, generați conform metodei gradienților conjugați, sunt ortogonali doi câte doi, deci, sunt liniar independenți, nu pot exista $n+1$ vectori reziduali nenuli. Rezultă că dacă calculele se fac exact, soluția exactă a sistemului (5.1) se găseşte în cel mult n paşi. Teoretic ar trebui ca vectorul rezidual $\bar{r}^{(n)}$ să fie nul, iar $\bar{x}^{(n)}$ să fie soluția exactă a sistemului (iar metoda una directă). În practică acest lucru nu se întâmplă, deoarece în determinarea vectorilor $\bar{r}^{(k)}$ intervin erori de calcul, care fac ca aceştia să nu formeze un sistem ortogonal. Deoarece în general $\bar{r}^{(n)} \neq \bar{0}$, continuăm să calculăm $\bar{r}^{(k)}$, $k > n$ până obținem $\left\|\bar{r}^{(k)}\right\| < \varepsilon$. Aceasta se justifică prin faptul că metoda gradienților conjugați este o metodă de relaxare prin care valoarea funcționalei $F(\bar{x})$ se micşorează la fiecare pas.

În cazul în care matricea coeficienților A este simetrică şi pozitiv definită, metoda converge pentru orice aproximație inițială $\bar{x}^{(0)}$.

Pentru implementare la fiecare iterație sunt necesare un produs matrice–vector, trei reînnoiri de vectori şi două produse scalare.

Metoda poate fi generalizată pentru rezolvarea sistemelor cu matrice nesingulare arbitrare. În acest caz, ținând cont că sistemul (5.1) este echivalent cu $A^T A\bar{x} = A^T\bar{b}$, a cărui matrice este simetrică şi pozitiv definită, se poate aplica metoda gradienților conjugați pentru rezolvarea ultimului sistem. Totuşi, se va evita această abordare atunci când matricea A a sistemului este slab condiționată.

III. Modul de desfăşurare a lucrării

♦ Cerințe față de îndeplinirea lucrării

În baza pseudocodurilor descrise în secțiunea II a lucrării, să se elaboreze un program care are drept destinație realizarea următoarelor etape:

1. Generarea de către utilizator a SEAL inițial de forma (5.1), în regim de dialog sau cu ajutorul unui generator de numere aleatoare, astfel încât să se cunoască soluția acestuia, iar matricea coeficienților A să satisfacă una dintre condițiile:

   a) A este simetrică; b) A este tridiagonală; c) A este nesingulară.
2. Rezolvarea SEAL obținut printr-o metodă directă, cât şi printr-o metodă iterativă.
3. Vizualizarea pe ecran sau stocarea într-un fişier a datelor intermediare în procesul de execuție a programului.

Parametrul de intrare al programului este dimensiunea n a sistemului. După preluarea acestui parametru, programul generează (interactiv sau aleator) un sistem de n ecuații şi n necunoscute, îl rezolvă şi afişează: eroarea absolută, eroarea relativă şi reziduul rezultatului, calculate fiecare prin formula normei euclidiene. Se afişează şi timpul de calcul.

♦ Chestiuni de studiat

1. Cum se pot verifica soluțiile obținute în urma rezolvării unui sistem de ecuații? Cum s-ar putea evalua eroarea cu care s-au determinat necunoscutele? Cât de bine este evaluată eroarea reală prin norma reziduului?
2. Dați exemplu de un sistem, cu matricea A nesingulară, care nu poate fi rezolvat prin a.e.G fără pivotare.
3. Să se facă o analiză experimentală a efectului diferitelor strategii de pivotare asupra erorii cu care se obține soluția, dar şi asupra timpului de calcul.
4. Să se compare valorile teoretice ale ordinului de complexitate a algoritmilor direcți elaborați cu valorile obținute în urma experimentărilor.
5. Să se facă o analiză experimentală a dependenței timpului de calcul, precum şi a normelor reziduului, în funcție de dimensiunea n a sistemului.
6. Să se facă o comparație între metodele directe şi cele iterative.
7. Există oare sisteme de ecuații, nedominant diagonale, care se pot rezolva totuşi prin metoda Gauss-Seidel, deşi metoda Jacobi nu este convergentă?
8. Care este explicația că inversând ordinea a două ecuații într-un sistem, metodele iterative ar putea să nu conveargă?

LUCRAREA DE LABORATOR №6
METODA LUI RUNGE-KUTTA
DE REZOLVARE A PROBLEMEI LUI CAUCHY

6.I. Scopul lucrării

Studiul unei familii de algoritmi numerici de rezolvare aproximativă a problemei lui Cauchy pentru ecuația diferențială ordinară de ordinul I şi implementarea acestora într-un limbaj de programare de nivel înalt.

6.II. Prezentarea teoretică a lucrării

♦ Formularea problemei

Considerăm problema lui Cauchy (problema cu valori inițiale) ataşată ecuației diferențiale ordinare de ordinul I

$$\begin{cases} y'(x) = f(x, y(x)), \ x \in [x_0; x_f] \subset \mathbb{R}, \\ y(x_0) = y_0 \ \ (y(x) \in C^1[x_0; x_f]), \end{cases} \tag{6.1}$$

în care $f(x, y(x))$ este o funcție cu proprietăți ce asigură existența şi unicitatea soluției în intervalul precizat. Să se determine soluția aproximativă numerică a problemei (6.1) cu exactitatea dată $\varepsilon > 0$, utilizând în acest scop metoda lui Runge-Kutta.

Problema (6.1) admite soluție unică $y^*(x)$ într-o vecinătate a punctului x_0 dacă într-un domeniu mărginit $D \subset [x_0; x_f] \times \mathbb{R}$ funcția $f(x, y)$ satisface următoarele condiții [2, p.116; 4, p.192; 5; 9; 12; 18]:

I) $f(x, y)$ este continuă în raport cu prima variabilă (pentru $x \in [x_0; x_f]$);

II) $f(x, y)$ satisface condiția lui Lipschitz în raport cu a doua variabilă y pentru orice $x \in [x_0; x_f]$:

$$\left| f(x, y_2) - f(x, y_1) \right| \le L \left| y_2 - y_1 \right|, \ \forall (x, y_1), (x, y_2) \in D \ \ (L > 0). \tag{6.2}$$

Dacă $\dfrac{\partial f(x, y)}{\partial y}$ este continuă în raport cu variabila y şi este mărginită pe D, atunci $f(x, y)$ satisface condiția (6.2).

Întrucât soluția exactă a problemei (6.1) se poate obține, prin tehnici clasice, numai pentru cazuri particulare, se justifică necesitatea recurgerii la metode aproximative. Se face distincție între metode de aproximare analitice şi metode numerice [9; 12; 18]. În cadrul primei categorii se încearcă să se găsească aproximație $y_{apr}(x)$ a soluției exacte $y^*(x)$, valabilă pentru orice $x \in [x_0; x_f]$. În cazul metodelor numerice, se încearcă să se găsească aproximații ale lui $y^*(x_i)$ pe o rețea de puncte $x_i \in [x_0; x_f]$, $i = 1, 2, \ldots$.

♦ Descrierea algoritmilor

Metoda monopas pentru rezolvarea numerică aproximativă a problemei (6.1) constă în generarea unei rețele de puncte x_i, $i = \overline{0, n}$, de pe intervalul $[x_0; x_f]$, posibil la distanțe diferite $h_i = x_i - x_{i-1}$, apoi, pornind de la condiția inițială $y(x_0) = y_0$, se calculează aproximația y_{i+1} a soluției exacte în punctul x_{i+1}, folosind numai aproximația în punctul precedent x_i $(i = 0, 1, \ldots)$. Astfel, avem o relație de recurență de forma $y_{i+1} = F(f, h_i, x_i, y_i)$, $i = 0, 1, \ldots, n-1$, care permite calculul succesiv al soluției numerice $y_1, \ldots, y_n$ în

toate nodurile rețelei de discretizare. Vom considera că soluția numerică $\{y_i\}_{i=1}^n$ este calculată cu exactitatea ε dacă se îndeplineşte inegalitatea $\left|y^*(x_i)-y_i\right|<\varepsilon,\ i=\overline{1,n}$.

Pentru a descrie o metodă monopas este suficient să arătăm cum se trece de la punctul (x_i,y_i) la punctul următor (x_i+h_i,y_{i+1}). Ne vom referi la aceasta ca fiind descrierea locală a unei metode monopas.

Forma generală a schemelor de calcul explicite de tip Runge-Kutta în p stadii se descrie prin formulele următoare [3-5; 9; 12; 18]:

$$y_{i+1}=y_i+\Delta y_i,\ \ \Delta y_i=\sum_{m=1}^{p}\alpha_m k_m,\ \ i=\overline{0,n-1}, \tag{6.3}$$

$$k_1=h\cdot f(x_i,y_i),\quad k_q=h\cdot f\left(x_i+\beta_q h,y_i+\sum_{j=1}^{q-1}c_{qj}k_j\right),\ q=\overline{2,p}. \tag{6.4}$$

E de menționat că în cazul dat punctele x_i se află la distanțe egale cu h. Coeficienții α_m, β_q şi c_{qj} ($m=\overline{1,p}$, $q=\overline{2,p}$, $j=\overline{1,q-1}$) se determină dezvoltând în serie Taylor membrul drept al relației (6.3) şi soluția exactă $y^*(x)$ în vecinătatea punctului x_i, şi identificând coeficienții expresiilor obținute. Din motive de simplitate a formulelor se impun condițiile $\beta_q=\sum_{j=1}^{q-1}c_{qj}$, $q=\overline{2,p}$, $\sum_{m=1}^{p}\alpha_m=1$.

Schema Runge-Kutta (6.3)-(6.4) este de ordinul $s\in\mathbb{N}$, dacă eroarea de trunchiere pentru un pas al rețelei $\rho_{i+1}:=y^*(x_i+h)-y_{i+1}$ satisface condiția [9; 12; 18]

$$\rho_{i+1}=ch^{s+1}\ \ (c=const), \tag{6.5}$$

adică dacă seria Taylor pentru soluția exactă $y^*(x_i+h)$ coincide cu cea pentru y_{i+1} până la termenul ce conține pe h^s inclusiv. Dacă s este ordinul schemei explicite în p stadii (6.3)-(6.4), atunci $s=p$, $p=\overline{1,4}$ [9; 12; 18]. În continuare vom considera scheme explicite de tip Runge-Kutta în $p\le 4$ stadii.

Pentru a determina aproximația y_{i+1} ($i=\overline{0,n-1}$) conform formulelor (6.3)-(6.4) se calculează valorile funcției $f(x,y)$ în puncte intermediare ale intervalului $[x_i;x_{i+1}]$, numărul de puncte fiind legat direct de ordinul p al metodei.

Indicăm câteva scheme de calcul frecvent utilizate, care se obțin pentru valori concrete ale coeficienților formulelor (6.3)-(6.4) [5; 9; 12; 18]:

A) $\Delta y_i=(k_1+k_2)/2,\ k_1=h\cdot f(x_i,y_i),\ k_2=h\cdot f(x_i+h,y_i+k_1);$ *B)* $\Delta y_i=k_2,\ k_1=h\cdot f(x_i,y_i),\ k_2=h\cdot f(x_i+h/2,y_i+k_1/2);$ *C)* $\Delta y_i=(k_1+3k_2)/4,\ k_1=h\cdot f(x_i,y_i),\ k_2=h\cdot f(x_i+2h/3,y_i+2k_1/3);$
D) $\Delta y_i=(k_1+4k_2+k_3)/6,\ k_1=h\cdot f(x_i,y_i),\ k_2=h\cdot f(x_i+h/2,y_i+k_1/2),\ k_3=h\cdot f(x_i+h,y_i-k_1+2k_2);$ *E)* $\Delta y_i=(k_1+3k_3)/4,\ k_1=h\cdot f(x_i,y_i),\ k_2=h\cdot f(x_i+h/3,y_i+k_1/3),\ k_3=h\cdot f(x_i+2h/3,y_i+2k_2/3);$ *F)* $\Delta y_i=(2k_1+3k_2+4k_3)/9,\ k_1=h\cdot f(x_i,y_i),\ k_2=h\cdot f(x_i+h/2,y_i+k_1/2),\ k_3=h\cdot f(x_i+3h/4,y_i+3k_2/4);$

$$G)\quad \Delta y_i=(k_1+2k_2+2k_3+k_4)/6,\ \ k_1=h\cdot f(x_i,y_i),\ \ k_2=h\cdot f(x_i+h/2,y_i+k_1/2),$$
$$k_3=h\cdot f(x_i+h/2,y_i+k_2/2),\ \ k_4=h\cdot f(x_i+h,y_i+k_3);$$
$$H)\quad \Delta y_i=(k_1+4k_3+k_4)/6,\ \ k_1=h\cdot f(x_i,y_i),\ \ k_2=h\cdot f(x_i+h/4,y_i+k_1/4),$$
$$k_3=h\cdot f(x_i+h/2,y_i+k_2/2),\ \ k_4=h\cdot f(x_i+h,y_i+k_1-2k_2+2k_3);$$
$$I)\quad \Delta y_i=(k_1+3k_2+3k_3+k_4)/8,\ \ k_1=h\cdot f(x_i,y_i),\ \ k_2=h\cdot f(x_i+h/3,y_i+k_1/3),$$
$$k_3=h\cdot f(x_i+2h/3,y_i-k_1/3+k_2),\ \ k_4=h\cdot f(x_i+h,y_i+k_1-k_2+k_3).$$

♦ **Monitorizarea erorilor locale de trunchiere pe parcursul procesului de integrare**

Întrucât în practică evaluarea constantei c din relația (6.5) este dificilă, monitorizarea erorilor locale de trunchiere pe parcursul procesului de integrare a ecuației diferențiale se face în baza unor relații aproximative asimptotice. Acestea din urmă utilizează două aproximații ale soluției în acelaşi punct.

- *Estimarea erorii locale în baza principiului lui Runge* [5; 9; 12; 18]

Estimarea erorii de trunchiere pentru un pas la formulele lui Runge-Kutta (6.3)-(6.4) se poate face repetând calculul cu pas modificat. Spre exemplu, pentru a verifica dacă pasul h este suficient de mic pentru asigurarea unei precizii corespunzătoare, calculăm valoarea aproximativă a soluției în punctul x_{i+1}, avansând doi paşi consecutivi, începând cu punctul x_i, conform aceleiaşi formule cu pasul $h/2$, apoi se reface calculul aproximației în punctul x_{i+1} cu pasul dublu h. Notăm prin $y_{i+1}^{h/2}$ şi, respectiv, y_{i+1}^{h} aproximațiile soluției $y^*(x)$ în x_i+h, obținute cu paşii de integrare $h/2$, respectiv h. Atunci o estimare a erorii locale de trunchiere poate fi obținută prin relația [9; 12; 15-16; 18; 25]

$$y(x_{i+1})-y_{i+1}^{h/2}\simeq\left(y_{i+1}^{h/2}-y_{i+1}^{h}\right)\Big/\left(2^p-1\right),\qquad (6.6)$$

care are ordinul de exactitate $O(h^{p+1})$ în raport cu pasul diviziunii (p - ordinul formulei lui Runge-Kutta utilizate).

- *Estimarea erorii locale utilizând principiul metodelor scufundate (principiul lui Fehlberg)* [9; 12; 18]

O cale mai eficientă (în sensul economiei resurselor de calcul utilizate) de a obține informații privitoare la precizia calculelor constă în utilizarea unei perechi de formule Runge-Kutta scufundate (imbricate), adică a două formule Runge-Kutta de ordine p şi p^*, cu $p^*>p$. Conform acestor formule calculele se fac cu acelaşi pas h.

Să considerăm schema de calcul explicită de tip Runge-Kutta în p stadii, ai cărei coeficienți sunt α_m,β_q,c_{qj} $(m=\overline{1,p},\ q=\overline{2,p},\ j=\overline{1,q-1})$. Alegem un proces similar în p^* stadii, cu $p^*>p$, astfel încât coeficienții acestuia satisfac relațiile $\beta_q^*=\beta_q,\ c_{qj}^*=c_{qj},\ q=\overline{2,p}$ (de exemplu, această proprietate vor satisface schemele B şi D, B şi F, B şi G definite mai sus). Atunci $k_q^*=k_q,\ q=\overline{1,p}$. Fie $y_{i+1}^{p^*}$ şi y_{i+1}^{p} valorile obținute în urma aplicării schemelor în p^* şi, respectiv, p stadii. Atunci eroarea locală $\rho_{i+1}^{p}:=y^*(x_i+h)-y_{i+1}^{p}$ a formulei în p stadii poate fi estimată prin relația [12; 18, p.23]

$$\rho_{i+1}^{p}\simeq y_{i+1}^{p^*}-y_{i+1}^{p}=\sum_{m=1}^{p^*}\gamma_m k_m^*,\ \ \gamma_m=\begin{cases}\alpha_m^*-\alpha_m, & m=\overline{1,p}\\ \alpha_m^*, & m=\overline{p+1,p^*}\end{cases}.\qquad (6.7)$$

Valoarea aproximativă se obține cu o eroare de ordinul $O(h^{p+1})$. În calitate de aproximație a soluției în punctul $x_{i+1} := x_i + h$ se poate considera valoarea $y_{i+1}^{p^*}$.

♦ **Algoritmi cu controlul automat al pasului**

Ținând cont de estimațiile (6.6) sau (6.7) pentru eroarea locală de trunchiere a metodei, se poate dirija procedura de selecție a mărimii pasului de integrare h_i în procesul de calcul astfel, încât să se îndeplinească inegalitatea $\left|y^*(x_i + h_i) - y_{i+1}\right| < \varepsilon$ în toate punctele rețelei de discretizare.

Algoritmul 1

Acest algoritm utilizează procedeul înjumătățirii pasului de integrare [9; 12; 15-16; 25-26].

După aceleaşi formule calculăm valorile aproximative ale soluției în punctele x_i de pe $[x_0; x_f]$ cu pasul h, apoi în punctele x_{2i} cu pasul $h/2$. Notăm aceste valori respectiv prin y_i^h şi $y_i^{h/2}$. Considerăm că exactitatea s-a atins, dacă în punctele comune se îndeplinesc condițiile:

$$Dy_i = \left|y_i^h - y_i^{h/2}\right| < \varepsilon_1,\ \varepsilon_1 = (2^p - 1)\varepsilon .$$

În caz contrar, se continuă divizarea pasului, adică se fac calculele cu paşii $h/2$ şi $h/4$ etc. Se recomandă, totuşi, să se limiteze divizarea pasului în cazul în care nu se atinge exactitatea într-un număr rezonabil de iterații.

Algoritmul 2

Algoritmul 1 poate fi optimizat (astfel încât să diminueze efortul de calcul, păstrând totodată exactitatea dorită a soluției aproximative) dacă se asigură alegerea mărimii pasului în procesul de calcul în dependență de caracterul variației soluției ecuației diferențiale. Vom utiliza procedeul înjumătățirii sau dublării pasului de integrare [18, p.27].

Fie ρ_{i+1} estimația erorii de trunchiere a metodei (calculată în baza relațiilor (6.6) sau (6.7)) ce apare în calculul aproximației y_{i+1}^h în punctul $x_i + h$. Dacă $\left|\rho_{i+1}\right| \geq \varepsilon$, atunci valoarea y_{i+1}^h nu satisface exactitatea dorită. Punctul obținut $x_i + h$ şi valoarea y_{i+1}^h se exclud din studiu. Se pune $h^{(1)} = h/2$ şi din nou, cu aceeaşi formulă Runge-Kutta (dar cu pasul $h^{(1)}$) se calculează o nouă aproximație $y_{i+1}^{h^{(1)}}$ în punctul $x_i + h^{(1)}$. Fie $\rho_{i+1}^{(1)}$ - estimația erorii de trunchiere ce apare în calculul lui $y_{i+1}^{h^{(1)}}$. Dacă din nou $\left|\rho_{i+1}^{(1)}\right| \geq \varepsilon$, atunci punctul $x_i + h^{(1)}$ şi valoarea $y_{i+1}^{h^{(1)}}$ se exclud din studiu, iar pasul se înjumătățeşte $h^{(2)} = h^{(1)}/2$ şi calculele se repetă. Procesul continuă până atunci când pentru o valoare a pasului (o vom nota prin h_i) estimația erorii satisface $\left|\rho_{i+1}\right| < \varepsilon$. În acest caz se consideră că soluția ecuației diferențiale a fost prelungită până în punctul $x_{i+1} = x_i + h_i$. În continuare ecuația diferențială se integrează, începând cu punctul x_{i+1} şi pasul h_{i+1}, care se alege în modul următor:

Dacă estimația erorii la pasul $h_i = x_{i+1} - x_i$ satisface inegalitatea $\left|\rho_{i+1}\right| \leq \varepsilon/K$ (constanta K se consideră egală cu 2^{p+1}, unde p este ordinul metodei), atunci pasul de integrare se dublează $h_{i+1} = 2h_i$. Dacă însă $\varepsilon/K < \left|\rho_{i+1}\right| < \varepsilon$, atunci pasul de integrare rămâne acelaşi $h_{i+1} = h_i$.

Astfel pe porțiunile de variație ale argumentului x, pe care se atinge o exactitate înaltă a soluției aproximative, pasul de integrare poate să ia valori mai mari, iar acolo unde exactitatea nu se atinge - descreşte până la o mărime ce asigură atingerea exactității.

Algoritmul 3

Acest algoritm utilizează procedeul pasului maxim admisibil în limita exactității impuse [18, p.29].

Fie ρ_{i+1} o estimație a erorii de trunchiere a metodei ce apare în calculul aproximației y_{i+1}^h în punctul $x_i + h$.

Dacă $|\rho_{i+1}| \geq \varepsilon$, atunci aproximația y_{i+1}^h nu atinge exactitatea dorită pentru aşa mărime a pasului h, drept urmare, y_{i+1}^h şi punctul $x_i + h$ se exclud din studiu. Se alege o nouă mărime a pasului, dar nu prin metoda înjumătățirii, ca în algoritmul 2, ci cu ajutorul relației

$$h_\varepsilon = \alpha h, \;\; \alpha = \sqrt[p+1]{\varepsilon / |\rho_{i+1}|} \;\; (p \text{ - ordinul metodei}). \tag{6.8}$$

Întrucât $|\rho_{i+1}| \geq \varepsilon$ avem $\alpha < 1$ şi $h_\varepsilon < h$. Prin formula Runge-Kutta cu pasul h_ε se calculează aproximația $y_{i+1}^{h_\varepsilon}$ în punctul $x_i + h_\varepsilon$. Pentru eroarea de trunchiere la acest pas avem $|\rho_{i+1}^{(\varepsilon)}| \approx \varepsilon$ [18, p.30].

Dacă însă estimația erorii în punctul $x_i + h$ satisface condiția $|\rho_{i+1}| < \varepsilon$, atunci se consideră că aproximația obținută y_{i+1}^h satisface exactitatea dorită, iar valoarea $x_i + h$ se acceptă în calitate de nou nod al rețelei. Se calculează valoarea h_ε conform formulei (6.8).

La această etapă soluția ecuației diferențiale a fost prelungită până în punctul x_{i+1} ($x_{i+1} = x_i + h_\varepsilon$ dacă $|\rho_{i+1}| \geq \varepsilon$ şi $x_{i+1} = x_i + h$ dacă $|\rho_{i+1}| < \varepsilon$). În continuare ecuația diferențială se integrează, începând cu punctul x_{i+1}. În calitate de pas ce va fi utilizat pentru testarea preciziei aproximației în $x_{i+1} + h$ se consideră $h = h_\varepsilon$.

Se recomandă să se micşoreze puțin valoarea lui α, de exemplu $\alpha^* = 0.9\alpha$ şi, respectiv, valoarea pasului $h_\varepsilon = \alpha^* h$. Aceasta se face cu scopul de a evita acei paşi pentru care nu se atinge exactitatea cerută.

Remarcă. Spre deosebire de primul algoritm, ceilalți doi permit calculul soluției numerice în nodurile unei rețele neuniforme (în general vorbind), care nu este definită apriori, ci se construieşte automat în procesul de integrare numerică. În calitate de noduri ale rețelei se iau punctele segmentului de integrare, în care se îndeplineşte criteriul $|\rho_{i+1}| < \varepsilon$. În aceşti doi algoritmi nu este prevăzut modul în care se aproximează soluția în extremitatea din dreapta a intervalului de integrare. Această problemă necesită o abordare individuală.

Pseudocodul algoritmului

Vom da o variantă de pseudocod ce realizează calculul soluției numerice a problemei (6.1) cu exactitatea ε, utilizând schema de calcul G şi algoritmul 1.

Date de intrare:
$x0$ (xf) - (de tip real) limita inferioară (superioară) a intervalului de integrare a ecuației diferențiale;
$y0$ - (de tip real) – valoarea soluției în punctul inițial $x0$;

f - numele subprogramului-funcție (de tip real) cu ajutorul căruia se calculează valoarea funcției $f(x,y)$; n (de tip întreg) – numărul inițial de divizări ale intervalului $[x0; xf]$; ε (de tip real) – exactitatea de calcul a soluției numerice. Date de ieşire: xi - (de tip real) – punctul în care se calculează soluția aproximativă numerică; $yih2$ - (de tip real) - valoarea soluției problemei calculată în punctul xi cu exactitatea ε ; Dyi - (de tip real) – estimație a erorii aproximării lui $y^*(x)$ în punctul xi ; h - (de tip real) – valoarea pasului de integrare la care s-a atins exactitatea dorită; real $h, xi, yih, yih2, Dyi$

funcție $dlyi(h, xi, yi)$
{
real $h, xi, yi, k1, k2, k3, k4$
$k1 := h \cdot f(xi, yi)$; $k2 := h \cdot f(xi + h/2, yi + k1/2)$
$k3 := h \cdot f(xi + h/2, yi + k2/2)$; $k4 := h \cdot f(xi + h, yi + k3)$
returnează $(k1 + 2 \cdot k2 + 2 \cdot k3 + k4)/6$
}
$h := (xf - x0)/n$
execută
{
$xi := x0$; $yih := y0$; $yih2 := y0$; $h := 0.5h$
execută
{
$yih2 := yih2 + dlyi(h, xi, yih2)$; $yih2 := yih2 + dlyi(h, xi + h, yih2)$
$yih := yih + dlyi(2h, xi, yih)$; $xi := xi + 2h$
$Dyi := \lvert yih2 - yih \rvert$
scrie în fişier xi, $yih2$, Dyi, h
}
cât timp ($Dyi < \varepsilon$ şi $xi \le xf$)
}
cât timp $Dyi \ge \varepsilon$

Analiza algoritmilor

Algoritmii metodei Runge-Kutta analizați în lucrare sunt algoritmi expliciți monopas, adică la determinarea lui y_{i+1} utilizează informațiile referitoare numai la un singur punct anterior, corespunzător lui x_i. În raport cu algoritmul lui Euler [9; 12; 16; 18] aceştia asigură îmbunătățirea preciziei şi a erorii de trunchiere pe un pas de integrare. Pentru reducerea erorii, la determinarea lui y_{i+1}, $i = \overline{0, n-1}$, se calculează valorile funcției $f(x,y)$ din membrul drept al ecuației diferențiale într-un număr de puncte intermediare ale intervalului $[x_i; x_{i+1}]$.

Dacă se doreşte, în baza soluției numerice $\{y_i\}_{i=1}^{n}$ se poate obține prin interpolare aproximație $y_n(x)$, definită pe întreg intervalul $[x_0; x_f]$.

Calculul aproximației y_{i+1}, cunoscând y_i, necesită p evaluări ale lui $f(x,y)$. Efectuarea calculelor conform relației (6.6) implică $3p-1$ evaluări ale lui $f(x,y)$ (schema Runge-Kutta (6.3)-(6.4) se aplică de trei ori), iar conform relației (6.7) - p^* evaluări. Dacă p^* este apropiat de p, atunci, în cazul în care se utilizează relația (6.7) (comparativ cu relația (6.6)), scade esențial numărul de evaluări ale funcției

$f(x,y)$. De exemplu, pentru schemele B şi D, definite mai sus, aplicarea relației (6.7) necesită trei evaluări ale funcției $f(x,y)$, pe când aplicarea relației (6.6) – cinci evaluări.

Vom menționa încă o dată, că estimațiile (6.6) şi (6.7) sunt asimptotice, deci sunt adevărate pentru valori relativ mici ale pasului h.

E de remarcat faptul că pentru anumite valori ale pasului de integrare h poate să apară fenomenul de instabilitate numerică a algoritmilor (erorile se propagă în mod imprevizibil sau chiar nemărginit) [5; 9; 12; 18]. Pentru asigurarea stabilității numerice se poate proceda la reducerea "cu grijă" a valorii pasului de integrare h până la o valoare suficient de mică. Totodată, se va ține cont că reducerea valorii lui h determină creşterea considerabilă a volumului de calcule legate de evaluările funcției $f(x,y)$.

Pentru algoritmii numerici analizați mai sus, erorile de trunchiere sunt:

- de ordinul de mărime al lui h^3 pentru versiunile A,B şi C ale algoritmilor;
- de ordinul de mărime al lui h^4 pentru versiunile D,E şi F ale algoritmilor;
- de ordinul de mărime al lui h^5 pentru versiunile G,H şi I ale algoritmilor.

Erorile de rotunjire depind de posibilitățile echipamentului de calcul pe care sunt implementați algoritmii. Aceste erori pot fi diminuate dacă se utilizează modul de lucru în *dublă precizie*. Cu toate acestea, erorile de rotunjire cresc pe măsura creşterii numărului de paşi de integrare datorită propagării erorilor de la un pas la altul. Prin urmare, pasul de integrare poate fi scăzut (în vederea micşorării erorii de trunchiere) numai atâta timp, cât eroarea de rotunjire rămâne suficient de mică.

Pentru memorarea vectorului $\overline{y}=\{y(x_i)\}$ sunt necesare n locații de memorie. Însă, în scopul eliminării acestui consum de memorie, soluția poate fi afişată (sau salvată pe disc), imediat după ce a fost calculată. Timpul de calcul al soluției numerice va depinde în mod esențial de structura funcției $f(x,y)$, precum şi de numărul de evaluări ale acesteia.

3.III. Modul de desfăşurare a lucrării

♦ **Cerințe față de îndeplinirea lucrării**

În baza algoritmilor descrişi în secțiunea II a lucrării, să se elaboreze un program care are drept obiectiv determinarea soluției numerice a probleme (6.1) cu exactitatea dată $\varepsilon>0$ sau cu cea mai bună exactitate posibilă (în cazul în care începe să se manifeste fenomenul de instabilitate numerică), utilizând:

- alegerea automată a pasului de integrare (algoritmii 1-3);
- diferite modalități de estimare a erorii de trunchiere (relațiile (6.6) şi (6.7)).

Să se testeze programul de calcul pe exemplele propuse la sfârşitul lucrării de laborator. Pentru fiecare dintre aceste exemple să se afle soluția exactă $y^*(x)$, soluția aproximativă sub formă numerică (utilizând unul dintre algoritmii Runge-Kutta $A-I$ descrişi mai sus), eroarea soluției aproximative, numărul de paşi efectuați, pasul mediu şi numărul total de evaluări ale părții drepte $f(x,y)$ a ecuației diferențiale. Să se reprezinte grafic variația erorii de calcul în funcție de valoarea pasului h.

Calculele încep cu pasul $h=0.1$, nodul final este $x_f=x_0+1$, iar exactitatea de calcul este $\varepsilon=10^{-4}$.

Să se extragă la tipar tabelul:

x_i	$y^*(x_i)$	y_i	Dy_i
…	…	…	…

x_i $(i=\overline{0,n})$ - nodurile rețelei obținute;

$y^*(x_i)$ $(i=\overline{0,n})$ - valorile soluției exacte $y^*(x)$ în punctele x_i;

y_i $(i=\overline{0,n})$ - valorile aproximative ale soluției $y^*(x)$ în punctele x_i;

Dy_i $(i=\overline{0,n})$ - eroarea locală de trunchiere în punctele x_i.

♦ **Chestiuni de studiat**

1. Să se facă o analiză experimentală a evoluției erorii de calcul în procesul de realizare a algoritmilor, precum şi a timpului de calcul al soluției numerice, în funcție de pasul de integrare şi de exactitatea impusă.
2. Să se facă o analiză comparativă a algoritmilor Runge-Kutta din punctul de vedere al efortului de calcul necesar obținerii unei exactități impuse a soluției.
3. Analizați deosebirea dintre stabilitatea unei ecuații diferențiale şi stabilitatea ei numerică.
4. Scrieți un algoritm pentru calculul integralei definite ale unei funcții aplicând metoda lui Runge-Kutta.

♦ **Probleme propuse spre rezolvare**

1. $y'=(3-4y)/2x,\quad y(1)=-4;$
2. $y'=x/y,\quad y(2)=-1;$
3. $y'=2-e^{-4x}-2y,\quad y(0)=1;$
4. $y'=y/x+1,\quad y(1)=1;$
5. $y'=4x+y,\quad y(0)=-3;$
6. $y'=y/x-2\ln x/x,\quad y(1)=1;$
7. $y'=-y/x+(x+1)e^x/x,\quad y(1)=e;$
8. $y'=y\cos x-\sin 2x,\quad y(0)=3;$
9. $y'=2y/(x+1)+(x+1)^3,\quad y(0)=0.5;$
10. $y'=-2xy+xe^{-x^2}\sin x,\quad y(0)=1;$
11. $y'=2y/(x+1)+e^x(x+1)^2,\quad y(0)=1;$
12. $y'=y/x+x\sin x,\quad y(\pi/2)=1;$
13. $y'=-2xy-2x^3,\quad y(1)=1/e;$
14. $y'=y/(x+1)+e^x(x+1),\quad y(0)=1;$
15. $y'=y/(x+2)+x^2+2x,\quad y(-1)=1.5;$
16. $y'=-ytgx+\cos^2 x,\quad y(\pi/4)=0.5;$
17. $y'=-y\cos x+0.5\sin 2x,\quad y(0)=0;$
18. $y'=yctgx+2x\sin x,\quad y(\pi/2)=0;$
19. $y'=2xy/(x^2+1)+x^2+1,\quad y(1)=3;$
20. $y'=-3y/x+2/x^3,\quad y(1)=1;$
21. $y'=y/x-12/x^3,\quad y(1)=4;$
22. $y'=-2y/x+x^3,\quad y(1)=-5/6;$
23. $y'=y\cos x+\sin 2x,\quad y(0)=-1;$
24. $y'=3x^2y+x^2(x^3+1)/3,\quad y(0)=0;$
25. $y'=-xy-x^3,\quad y(0)=3;$
26. $y'=4xy-4x^3,\quad y(0)=-0.5.$

REFERINŢE

1. Buzurniuc Ş., Moraru V. Informatica: elemente de calcul numeric. – Chişinău: Evrica, 2000. –116 p.
2. Dăneţ N. Analiza numerică cu aplicaţii rezolvate în Mathcad. – Bucureşti: Matrix Rom, 2002. – 247 p.
3. Mitran S., Berbente C., Zancu S. Metode numerice. – Bucureşti: Editura Tehnică, 1997. – 306 p.
4. Păltineanu G., Matei P., Trandafir R. Bazele analizei numerice. – Bucureşti: Printech, 2001. – 284 p.
5. Secrieru G.V., Secrieru I.V. Analiza numerică. – Chişinău: Ştiinţa, 1985. – 207 p. (cu alfabet chirilic)
6. Secrieru I., Râbacova G. Analiza numerică în probleme şi exemple. – Chişinău: CEP USM, 2003. – 78 p.
7. Botoşanu M., Secrieru I., Zavadschi V. Elemente de modelare şi analiză numerică. Informatica (material didactic pentru licee şi colegii). – Chişinău: Univers Pedagogic, 2006. – 131 p.
8. Burden R.L., Faires J.D. Numerical analysis (7 ed.). – Brooks Cole, 2001. – 841 p.
9. Butcher J.C. Numerical methods for ordinary differential equations. – Wiley, 2003. – 425 p.
10. Conte S.D., de Boor C. Elementary numerical analysis. An algorithmic approach (3rd ed.). – McGraw – Hill Book Company, 1980. – 432 p.
11. Golub G.H., Van Loan C.F. Matrix computations (3rd ed.). – Baltimore and London: The Johns Hopkins University Press, 1996. – 694 p.
12. Hairer E., Nørsett S.P., Wanner G. Solving ordinary differential equations I. Nonstiff problems (3rd ed.). – Springer-Verlag, 2008. – 528 p.
13. Kincaid D., Cheney W. Numerical analysis: mathematics of scientific computing. – Brooks Cole, 1991. – 690 p.
14. Saad Y. Iterative methods for sparse linear systems (2nd ed.). – SIAM, 2000. – 463 p.
15. Stoer J., Bulirsch R. Introduction to numerical analysis (2nd ed.). – Springer – Verlag, 1993. – 660 p.
16. Амосов А.А., Дубинский Ю.А., Копченова Н.В. Вычислительные методы для инженеров. – Москва: Высшая школа, 1994. – 544 с.
17. Арушанян О.Б. Практикум на ЭВМ по вычислительным методам. Численное решение нелинейных уравнений. – Москва: МГУ, 2003. – 37 с.
18. Арушанян О.Б., Залёткин С.Ф. Практикум на ЭВМ по вычислительным методам. Решение задачи Коши для обыкновенных дифференциальных уравнений одношаговыми разностными методами. – Москва: МГУ, 2002. – 51 с.
19. Баландин М.Ю., Шурина Э.П. Методы решения СЛАУ большой размерности. – Новосибирск: изд-во НГТУ, 2000. – 70 с.
20. Бахвалов Н.С., Жидков Н.П., Кобельков Г.М. Численные методы. – Москва: Наука, 1976. –630 с.
21. Березин И.С., Жидков Н.П. Методы вычислений, том I. – Москва: Наука, 1966. – 632 с.

22. Богачев К.Ю. Практикум на ЭВМ. Методы решения линейных систем и нахождения собственных значений. – Москва: изд-во МГУ, 1998. – 137 с.
23. Вержбицкий В.М. Основы численных методов: Учебник для вузов. – Москва: Высшая школа, 2002. – 840 с.
24. Демидович Б.П., Марон И.А. Основы вычислительной математики (3-е изд.). – Москва: Наука, 1966. – 664 с.
25. Копчёнова Н.В., Марон И.А. Вычислительная математика в примерах и задачах: Учебное пособие (2-е изд.). – Москва: Лань, 2008. – 368 с.
26. Костомаров Д.П., Фаворский А.П. Вводные лекции по численным методам: Учебное пособие. – Москва: Логос, 2004. – 184 с.
27. Самарский А.А., Гулин А.В. Численные методы: Учебное пособие для ВУЗов. – Москва: Наука, 1989. – 432 с.
28. Турчак Л.И., Плотников П.В. Основы численных методов, изд. второе. – Москва: Физматлит, 2003. – 304 с.